IBM Personal Systems Developer

A Publication of the IBM Developer Assistance Program
Fall 1991

- Spotlight on Micrografx®
- Computer Integrated Manufacturing
- Performance

IBM Personal Systems Developer

Table of Contents

The *IBM Personal Systems Developer* is published quarterly by IBM Software Developer Support, Internal Zip 2230, 1000 NW 51 Street, Boca Raton, Florida 33429. Phone (407) 982-6408, FAX (407) 443-4233. IBM employees and branch office customers can subscribe to the *Personal Systems Developer* through IBM Mechanicsburg's Systems Library Subscription Service (SLSS) using the *Developer's* order number, G362-0001. Others can subscribe by calling the publisher, Graphics Plus Inc., directly at (800) READ-OS2. Subscriptions (U.S. only) are $39.95 yearly. Questions, suggestions and article ideas should be sent to the Editor.

While back issues are not generally available, articles from the first seven issues of the *Personal Systems Developer* have been published in the *OS/2 Notebook: The Best of the IBM Personal Systems Developer*. This book can be bought at a local bookstore ($29.95) or by calling Microsoft Press at (800) MS-Press. Its Mechanicsburg order number is G362-0003-00.

Editor: Dick Conklin, IBM Software Developer Support. Publisher: Graphics Plus, Inc. 640 Knowlton Street, Bridgeport, CT 06608, Project Manager: Jo-Ann Radin-Campbell.

IBM Personal Systems Developer is published by the Entry Systems Division of International Business Machines Corporation, Boca Raton, Florida, U.S.A., Dick Conklin, Editor.

IBM believes the statements contained herein are accurate as of the date of publication of this document. **However, IBM hereby disclaims all warranties either expressed or implied, including without limitation any implied warranty of merchantability or fitness for a particular purpose. In no event will IBM be liable to you for any damages, including any lost profits, lost savings or other incidental or consequential damage arising out of the use or inability to use any information provided through this publication even if IBM has been advised of the possibility of such damages, or for any claim by any other party.**

Some states do not allow the limitation or exclusion of liability for incidental or consequential damages so the above limitation or exclusion may not apply to you.

This publication may contain technical inaccuracies or typographical errors. Also, illustrations contained here may show prototype equipment. Your system configuration may differ slightly.

This publication may contain articles by non-IBM authors. These articles represent the views of their authors. IBM does not endorse any non-IBM products that may be mentioned. Questions should be directed to the authors.

This information is not intended to be an assertion of future action. IBM expressly reserves the right to change or withdraw current products that may or may not have the same characteristics or codes listed in this publication. Should IBM modify its products in a way that may affect the information contained in this publication, IBM assumes no obligation whatever to inform any user of the modification.

It is possible that this material may contain reference to, or information about, IBM products (machines and programs), programming or services that are not announced in your country. Such references or information must not be construed to mean that IBM intends to announce such products, programming or services in your country.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever.

All specifications are subject to change without notice.

To correspond with the *IBM Personal Systems Developer*, please write to the Editor at IBM Corporation, Internal Zip 2230, P.O. Box 1328, Boca Raton, FL 33429-1328.

Trademarks

Editor's Comments

In this issue we'll cover a broad range of OS/2®'s capabilities, from something as fundamental as controlling a printer to running a real-time manufacturing plant.

First we will Spotlight the versatile Micrografx® Corporation, a pioneer on the PC's graphical user interface frontier. The people at Micrografx are involved in several OS/2 version 2.0 projects, including porting tools for applications and device drivers, a new graphics application, and the PM graphics engine.

We are introducing two new sections in this issue. The first is *Performance*. OS/2 benefits performance tuning in many ways, but a few good tools can sure help. We'll show you a program that you can use for high-resolution timing measurements — without the need for special hardware. Have you developed a large database application? We've got a cookbook of performance tips and techniques for you! We'll also introduce a new IBM product for OS/2 system performance management: System Performance Monitor/2™.

The other new department is *Manufacturing Applications*. Don't just think of OS/2 as an operating system for the office — today you'll find it on the shop floor too. Computer Integrated Manufacturing (CIM) applications written to the Distributed Automation Edition™ (DAE) application enabler include PlantWorks™ and the Paperless Manufacturing Workplace™ (PMW).

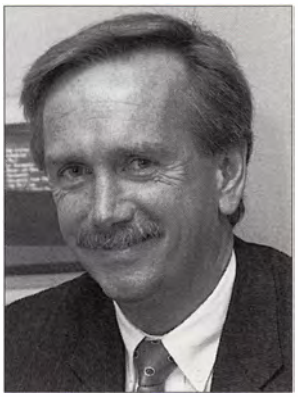
For those of you who are new to OS/2 or just need a little brushing-up on the fundamentals, we've got articles on the basics of multi-threading and on using the OS/2 Print Manager™. For the more advanced developer, there's IBM's Transaction Enabler (TxE), Arcadia's CUA Workbench™, and Periscope's® device driver debugger. For corporate developers, there's the SAA Delivery Manager™ for the administration, delivery and installation of applications on host-attached workstations.

At last year's Fall COMDEX™, IBM's Future Vision exhibit showed off an advanced, multiple-drive LAN server. Special software ran the system as a large-capacity, high-performance, fault-tolerant disk array. During the demonstration, a drive was removed, and all of the data was still online and available. That software is now available from Integra Technologies.

The Integrated Services Digital Network (ISDN) promises low-cost, high-speed communications over digital phone lines. Our ISDN team at the Austin Programming Center examines ISDN's hardware requirements and application potential.

In our Summer 1991 issue (*LAN Application Certification Support*, page 101), we told you how you can obtain more exposure for your OS/2 application. If you would like more information on this program, please call the author directly: **Art Borrego, (512) 823-3282**.

Don't forget to tell us about your OS/2 2.0 success stories and applications. *Our fax number is* ***(407) 443-4233***. In our next issue we'll revisit version 2.0 and some of its newer features.

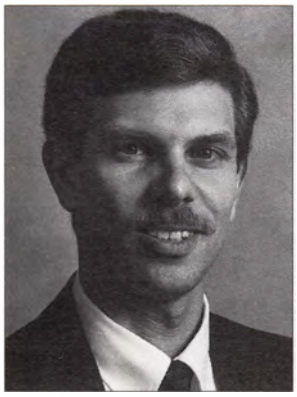


Dick Conklin

Dick Conklin

Developer Assistance Program

Developer Assistance Program Update



Joe Carusillo

by Joe Carusillo

OS/2® 2.0 EARLY CODE PROGRAMS

The Developer Assistance Program is running two successful OS/2 early code programs. We have filled over 1,500 requests for copies of OS/2 2.0 early code through the OS/2 2.0 Compatibility Test Program. As a result of this sucessful partnership with the software developer community, literally thousands of existing DOS®, Windows®, and OS/2 16-bit applications have been tested under OS/2 2.0. The OS/2 2.0 32-Bit Expedite Program has been equally successful. Under this program, we have received commitments from over 400 developers to produce new 32-bit specific products that will exploit the new capabilities of OS/2. For more information about the 32-Bit Expedite program please contact us.

FALL COMDEX™

If you are planning to attend this year's Fall COMDEX, make sure that you visit the IBM exhibits. Both are in East Hall. As in past years, there will be a separate exhibit (booth #728) devoted exclusively to showing software developer products that support OS/2. The focus for this year is to showcase a large number of the new 32-bit products that are being developed for OS/2 2.0. The Main IBM® exhibit (booth #1158) will feature demonstrations of, and presentations on, OS/2 2.0 and the entire IBM Personal Systems product line. The DAP information area will also be part of this exhibit. We will be providing information about the DAP, our services, and how to enroll. Again, I encourage you to stop by and see us if you are at the show.

SERVICES UPDATE

The services of the DAP include the IBMLink technical support system, ESDTOOLS, the OS/2 Information Exchange, and the Software Discount and Hardware Rebate programs. Last September, IBM announced three new communications packages that provide greatly improved performance and usability over the existing emulator that had been provided with each new IBMLink sign-on. The three new products are INPCS-High Performance Option Version 1.1, INPCS Entry Level Version 2.1, and IBM Personal Communications 3270 Version 2.0. All of these products have been added to the DAP Software Discount program and are available at forty percent off list price. All have been compatibility tested with OS/2 2.0 and will run unchanged. Also, the additional charge for 9600 baud service was dropped at that time.

If you haven't been keeping an eye on ESDTOOLS lately you need to. The number of utilities and demonstrations of tools keeps growing. There are now over 60 IBM authored tools and utilities and ten demos of commercial tools in the catalog. Look for new 32-bit versions of these products to start appearing over the next few months.

Since the last issue of the *Developer*, we have announced and made available an interesting special offering. This package is an IBM PS/2 Kanji keyboard and a set of software that allows developers to adapt their existing DOS

products to support the Japanese market. We have had a good response to this offering and will continue to offer it through year-end.

Other special offerings in the works are a set of hardware/software combinations to help developers accelerate their support of IBM's Multimedia products. Look for an announcement of this soon.

The OS/2 Information Exchange on CompuServe® has been a run-away success. Almost since the day it was first made available, it has been one of the most used services on CompuServe. If you haven't taken the time to look at it yet, I strongly encourage you to do so. There are 14 sections ranging from novice information to detailed developer information. I have spoken to a number of people that say that it is a must if you want to keep up on what's going on with OS/2.

Last but certainly not least, I'd like to update you on our Migration Workshops. We are rapidly updating the existing workshops to fully support OS/2 2.0 and a number of other new topics. Currently we have piloted and begun production of the DOS to OS/2 32-Bit Workshop and our new 16- to 32-Bit PM Workshop. Next-up is a new version of the Windows to PM Workshop that uses a beta version of the new IBM Software Migration Kit for Windows and a new OS/2 Database Performance Workshop. Both of these will also be piloted this year. Starting in 1992, the Migration Workshops will become a fee offering. The charge for a seat will be $2,000.00. For more information on Migration Workshops you can contact us at (407) 982-7920.

ELIGIBILITY AND ENROLLMENT

The IBM Developer Assistance Program is for software developers working on products for commercial release. There is no charge for membership. Participation is open to developers who:

- Develop products that support IBM OS/2 and/or IBM DOS
- Are currently marketing a PC software product

If you are not currently marketing a product, you can submit a non-confidential, detailed business plan showing marketing and development activities and schedules. IBM reserves the right to accept or reject an application based on this business plan. If you do not market the products you develop, you can contact us about a special consultant membership.

To request an application form or ask questions about the program, contact us at:

IBM Corporation
Developer Assistance Program
Internal Zip 2230
P.O. Box 1328
Boca Raton, FL 33429-1328
(407) 982-6408

Joe Carusillo, *IBM Personal Systems, 1000 NW 51st Street, Boca Raton, FL 33429. Mr. Carusillo is the manager of Software Developer Programs which includes the Developer Assistance Program. He started with IBM Entry Systems in 1982 and has worked on PC software since that time. He has a BS in Computer Systems Engineering from the Columbia University, School of Engineering and Applied Science, in the City of New York.*

Spotlight on Micrografx

Enabling Technologies for OS/2 2.0

Micrografx is at the forefront of PC graphics technology

by Peter Goldstein

Micrografx® has been at the forefront of PC graphics technology since its 1982 release of PC Draw™, the first drawing application available for the PC. The company is best known for its Windows®-based graphics applications: Micrografx Designer,™ Graph Plus,™ Micrografx Draw Plus,™ and Charisma.™ In fact, Micrografx was the first to ship an application for the Windows graphical environment.

Micrografx also has been involved in system level development. Its projects include Windows device driver development, a rich graphics API called MGXlib, and the creation of Micrografx Mirrors™ for porting Windows applications to OS/2® PM.

Micrografx currently is completing work on a number of enhancements under contract to IBM. These will be incorporated into the forthcoming OS/2 2.0. This issue's Spotlight will detail these enhancements, as well as provide an inside look at this most innovative company.

MICROGRAFX' INVOLVEMENT IN OS/2 2.0

Micrografx is working on four distinct projects that will be incorporated into OS/2 2.0: Mirrors Toolkit, Oasis Toolkit, 32-Bit GRE, and PM Chart.

Mirrors Toolkit

This toolkit will allow applications written for the Windows 3.0 environment to be ported to OS/2 2.0 PM with a minimal investment in resources, and by programmers without much PM experience. Ported applications will look and act like native PM applications, and typically will run faster than under Windows 3.0. A 32-bit support DLL will be included with every copy of OS/2 2.0 to provide a high performance compatibility and 16- to 32-bit interface layer between the ported applications and OS/2 2.0.

Oasis Toolkit

This toolkit provides two services. First, it provides a porting layer for Windows 3.0 device drivers to OS/2 presentation drivers (e.g., printers, plotters, film recorders).

Micrografx Inc.

Address:	1303 E Arapaho Road Richardson, TX 75081 (214) 234-1769
Founded:	1982
Employees:	250
Sales Channels:	Authorized distributors throughout the U.S. and the world. Seven international subsidiaries (UK, France, Germany, Denmark, Canada, Australia, and Spain) supplement the distributors outside of the U.S.
Other Products:	Designer and Charisma both for Windows, Designer for OS/2
Developer Contact:	Michael Yancey, Manager of Developer Relations, 1-800-733-3729

Second, it includes a DDK that allows native PM presentation drivers to be created quickly, without having to interface directly with the GRE. The Micrografx Plotter, PaintJet, Film Recorder, and PostScript Windows device drivers will be ported using Oasis in order to verify the functionality of Oasis. This set of four Micrografx drivers covers the vector plotter, raster printer, raster camera, and PostScript device technologies, respectively. In fact, the Micrografx PaintJet Driver is shipping with OS/2 2.0.

32-bit GRE

Micrografx, in conjunction with IBM, is rewriting the PM GRE to the 32-bit environment and focusing their attention on the graphics algorithms. Written entirely in C, there have been many significant enhancements in execution speed and functionality.

PM Chart

This is a scaled-down version of Micrografx Charisma, a data-driven graphics program. It will be included as part of the base OS/2 2.0 operating system.

COMPANY BACKGROUND

Micrografx was started in 1982 by brothers George and Paul Grayson in Richardson, Texas. They wrote an application in Interpreted BASIC called PC Draw, which became the first graphics program to ship for the still young IBM PC. In 1984 they met with Bill Gates of Microsoft® and agreed to develop a new drawing program for the not-yet-released Windows environment. Microsoft Windows Draw/Micrografx In*A*Vision™ was shipped with runtime Windows version 0.52, even before Windows was available as a separate product.

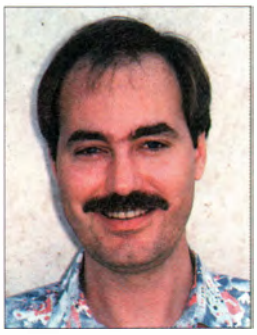
Micrografx worked closely with Microsoft during the early days of Windows, helping them to define Windows functionality along the way. Richard Merrick, Director of Systems Development, pointed out that like OS/2, "Windows didn't have adequate device support in the early days, which everybody forgets. Along the way we developed device drivers for different printers and plotters, either under contract or on our own."

Micrografx' printer drivers include the first HP Paintjet driver, a Postscript driver, a plotter driver, the Matrix Film Recorder driver, and the new Tektronix Color Phaser driver. "We were also instrumental in getting vector fonts put into Windows. We needed to scale fonts, and there was no outline font technology at the time."

Early OS/2 Development

"We went to both IBM and Microsoft when OS/2 was first being developed, explaining to them that it didn't make a whole lot of sense to make everybody recode their Windows applications," Merrick added. "We would have to double our staff, double our testing, double our support overhead, basically have a separate product line on two platforms. And for a company that was much smaller than we are now, it was very difficult to justify investing in a platform that wasn't guaranteeing a good return on investment. It would take a lot of development to reproduce all of the proprietary technology that we had in our Windows applications."

"When Microsoft asked us to beta test OS/2, we started by rewriting our simplest drawing application, Draw Plus, native to OS/2 PM. It was a complete redesign and coding effort, and after six months we had gotten about fifty percent of the way through...the easy fifty percent. Since there was not going to be any compatibility between Windows and OS/2 PM, we figured that there might be a chance we could write a portability layer that emulated Windows under PM. We researched the idea for a month or two and actually came back with a prototype that worked for Designer, our flagship Windows application. That became our PM strategy — providing as generic a piece of system software as possible, so we could use it to support all our applications as well." This porting tool became known as "Mirrors".



Richard Merrick

Merrick continued, "Development work on Mirrors began in mid-1988. I was brought on in early summer of 1989, almost immediately after the prototype was done, with the mission to grow the systems software side of the company. We immediately started getting phone calls from people all over the world asking to evaluate our toolkit to see if it would work for them. We had about 200 evaluators by January 1990. It was a who's who of independent and corporate software development. Some of them began formally licensing it in late 1989. We were getting feedback from all types of applications on how the product should work. We had people on-site working with us from developers like Gupta Technology and The Whitewater Group, vertical applications developers like PeopleSoft, and corporate customers like Boeing."

Other System Level Projects

Parallel to the early development of their Windows-to-OS/2 porting layer was another graphics toolkit, MGXlib. It is also known as the Micrografx Structured Graphics Toolkit. Its API provides about 450 low to high level graphics functions, dramatically reducing the amount of time it takes to provide sophisticated graphics routines in an application. Originally written for Windows, it has been ported to run under OS/2 PM with Micrografx' Mirrors Toolkit.

One of Micrografx' most recent projects was the development of the plot queue processor in OS/2 1.3, under contract to IBM Hursley. It is a complete reverse clipping engine that sits between an application and the plotter driver. On the screen, objects are drawn from the bottom layer up. Without clipping, overlapping objects will be layered on top of each other when output to a plotter, just as on the screen. On the plotter, overlapping colors will blend together, sometimes causing the paper to tear from ink saturation. In Micrografx' plot queue processor, objects are sent in reverse order through the clipping engine and broken down into simple polygons. Objects must be processed in this reverse order as clipping regions are defined relative to other overlapping objects. After clipping regions have been established, the remaining polygons are reconstructed into their original shapes, but without the clipped regions, and sent to the plotter.

"We got a whole lot of experience in understanding the PM Graphics Engine during this project," said Merrick. "We learned what the GRE was doing and why it acted in certain ways. We were having to hook out the area and path portions of the PM GRE, things no one had ever done. And, of course, there was no documentation on it. It was only with trial and error experimentation that we broke through it."

32-BIT PM GRAPHICS RENDERING ENGINE

One of the projects Micrografx currently is working on for incorporation into OS/2 2.0 is the rewriting of the PM GRE. Most of it is being written in portable C code, with a few performance-critical portions in 386 Assembler to support the Intel platform.

From the application developer's standpoint, the most significant difference will be a shift away from 16-bit integers in favor of 32-bit integers. David Raymer, Software Engineer, explained, "Because of the nature of the 386 machine, it prefers either a byte or a 32-bit value. So most of the things that the application programmer was used to being unsigned short values will become unsigned long values right away. Preexisting 16-bit executables will be able to be run via a simulated segment environment, but some minor changes will have to be made to new code in order to get it to run natively in the 32-bit environment. In short, new code will be different; old code will be able to run unmodified."

Ed Qualls

Ed Qualls, Software Engineer and Graphics Engine Project Lead, pointed out that, "If an application programmer mixes access of 16-bit and 32-bit data, return values, or constants, and doesn't get a warning from the compiler, the

programmer should watch out. The intricacies of compiler-generated sign extension or truncation will be trouble."

"The GPI interface in 1.x was essentially 32-bit to begin with, so far as things like x and y coordinates go," Ed added. "I think it would have been a major concern if we had to change such things in the GPI. But once inside the GRE, everything shifted down to 16-bit in OS/2 1.x. By staying 32-bit all the way through the GRE, we're getting more precision out of the math. We can now provide better integer-based emulation of floating point than OS/2 1.x afforded. Because we do the math using 64 bits, we end up with more precision than the original math, which did it in 16.16 or sometimes 32.16 bit chunks."

In the existing 16-bit GRE, if a vector is extended along the x-axis in an application, and then is rotated 'x' number of degrees, enough error will have been induced by the time 87.5 degrees of rotation has occurred that the system will think the line has been rotated a full 90 degrees. The 32-bit GRE will correct many inconsistencies like this.

Micrografx also is working on improving the performance of area fills, and on new entry points to support polygons. A closed path in OS/2 right now is just that. There is a lot of overhead in issuing a Beginpath, Endpath and many primitives in between. This is especially true when compared with Windows, where a polygon path is simply a data structure with points. Micrografx' addition of a new polygon entry point should improve display and printer performance while making it easier for application developers to code.

Charles Forsythe, Software Engineer, explained, "The system that supports polygons will also be able to fill with a clipping region. Currently you can't specify a clipping region on top of your fill path. This will provide a lot more flexibility, particularly for drivers that previously had to do their own clipping." Devices which can handle polygons natively can get the clipped fill area rebuilt into polygons. These are given back to the device using the new DevicePolygonSet device driver entry point. "This entry point is similar to GpiPolygons," Forsythe added, "but the polygons will not overlap or have crossing boundaries."

The presence of Polygons and DevicePolySet will dramatically improve the speed of area fills. Furthermore, Oasis will take advantage of the new entry points directly. They will reduce the amount of code necessary in Oasis while improving execution speed.

Micrografx also is adding its own high quality stretch bitmap handler to the new GRE. It will improve bitmap dithering while zooming in closer to the resolution of the bitmap itself. It also will retain much of the original quality as it reduces the bitmap. Merrick pointed out, "This is pretty unique stuff. Neither Microsoft nor anyone else really has it at this time, as it took us a long time to develop. It will be a real visual sign of improvement, as opposed to the more general enhancements in performance."

32-BIT MIRRORS TOOLKIT

Micrografx' most talked-about project, Mirrors, started off as an internal project to support its own efforts in porting existing Windows applications into the OS/2 PM environment. A technical overview of the Mirrors and Oasis Toolkits was printed in the Summer 1991 issue of the Developer, so we won't spend too much time repeating ourselves here. We talked with the Mirrors team at Micrografx to gain some more insight into the Mirrors technology and why it works the way it does.

Kurt Delimon

When Micrografx developed its first generation Mirrors toolkit to support its own migration projects, Microsoft licensed the technology and utilized it in its Windows Libraries for OS/2 (WLO). "Microsoft's strategy was to do everything dynamically with binary compatibility, where you could take a Windows application out of the box and it would load under PM," said Merrick. But as Kurt Delimon, Software Engineer and Mirrors Project Lead, pointed out, "There's no reason to convert many resources dynamically. It

makes more sense to convert them statically, binding them in as native PM resources in an application."

Delimon added, "Our goal has been to provide a higher performance application than direct binary compatibility can provide. The ultimate goal is better performance ported to OS/2 than with Windows under DOS. The only way we can achieve this goal is by not having binary compatibility hanging over our heads."

"About a year and a half ago, when we decided to rearchitect Mirrors, the theory was to do two things. First we had to keep it true to the Windows API, not just a kludge to make it work. Second was to implement Mirrors in terms of itself. All of the control classes, for example, are written using Windows calls. They don't use PM calls directly. The only place where we resolve the calls is at the Windows API level." Because the development of Mirrors required work on both Windows and OS/2 PM platforms, using its own technology allowed code to be maintained much more easily.

Making Windows Applications Look Like PM Applications

By focusing the utility of Mirrors on a static conversion of Windows applications, applications developers have more control over the operation of their programs under PM. Binary compatibility is useful only from a user's standpoint. It will allow them to migrate to OS/2 and still be able to run their Windows applications. But for the applications developer wanting to produce a native looking PM application, Mirrors offers much more with a minimum of effort.

"One of the things you run into with any type of porting layer is that you want to make the application fit into the environment as closely as possible," Delimon stressed. "But there are certain things which either don't apply or can't provide an equivalent mapping." Scroll bars are an excellent example of this dilemma. Under PM, the arrows become disabled and no message is sent when the top or bottom is reached. Under Windows, the scrollbar still will send messages when the lineup or linedown button is selected, even if the thumb is at the top or bottom of its range. If an application has used a scrollbar to simulate a spinner control, following the PM behavior of disabling the scroll buttons would break the application.

"The porting layer can't know what context the resources are used in," Delimon added, "so we're providing extensions to allow the applications developer to make decisions. There's a hook API that essentially lets you take your existing Windows application and make particular calls directly to PM. In the scroll bar example, a developer could make a function call that alters the behavior of the scroll bars based on whether he or she wanted them to work either as they do in PM or as they do in Windows. It is something that only the application can give, and that's obviously something you can't get from binary compatibility. Instead of just porting your application you can go in and actually tailor your application to the environment that you're running under."

Easing the Developer's Learning Curve

From a programmer's standpoint, Mirrors eases the migration to OS/2 PM. "One of the things I think people do when they come from Windows to PM is think in terms of the Windows API, trying to map it directly to PM, as opposed to looking at the broader scope of things," Delimon remarked.

"We've tried to make the learning curve as short as possible, so that Windows developers don't have to have this huge learning curve in front of them again. The point is, they've already invested their time and effort in the learning curve of Windows, so Mirrors should be able to come along and be something relatively easy for them to pick up in order to arrive at a PM application."

Jeff English, Software Engineer, went on to say, "I just finished porting our entire shipping version of Designer, its slide show, batch print program, and all the translators involved, a total of probably 18 executables and DLLs. The entire porting process of relinking and then getting the PM application to load only took about a week."

"Of course there is some up front work the first time you do the port," Merrick said. "In Windows there is a function named DOS3CALL. You basically write to this API and then, under DOS, it maps to the requested DOS interrupt. Under PM, it maps to the appropriate DOS API. So you've got some up front work to do to make your application host independent."

More Anomalies between Windows and PM

There are many differences between the way Windows and PM provide services. The job at Micrografx was to minimize the impact of these differences on the application developer, making the appropriate conversions in the porting layer.

One such adaptation occurs in hit testing. English explained, "There's a difference between the way that Windows does hit testing and the way that PM does hit testing. When you move the mouse over a window, the system sends a hit test message up through the window stack to see if that window can receive mouse messages. Windows starts at the topmost window to see if it is transparent or can receive mouse messages. PM starts from the bottom up. In PM if the first window says it is transparent, then none of its child windows will get a hit test message. You must have a parent window return that it can receive hit test messages if its children can receive them. When a parent window in a ported Windows application returns that it is transparent, we essentially change it around to tell PM that it can receive mouse messages. This way the hit testing continues on up the chain of child windows."

"Another difference is that in PM, the operation of the frame and client windows aren't coupled as tightly as under Windows. Under Windows, the client window receives nonclient messages for any activity involving the frame or frame controls. Under PM, the client window is treated as a separate entity and you have to subclass the frame window and its controls in order to be aware of any activity involving the frame. In Mirrors, we have to be conscious of this so that we can maintain proper support for the Windows applications."

PM CHART - OS/2 MINI APP PORTED WITH MIRRORS

Moving down the list of concurrent projects that Micrografx is working on for OS/2 2.0 brings us to PM Chart. It is a data-driven graphics program that will be included as part of the base OS/2 2.0 operating system. PM Chart basically is a scaled-down version of Charisma, Micrografx' full-featured graphing, charting, and drawing program for Windows. It reads and writes Micrografx' standard .DRW and .GRF file formats, thus maintaining compatibility with other Windows and OS/2 PM applications.

The basic operation of PM Chart calls for data to be entered into its worksheet or imported from a file, via the Clipboard, or a DDE link. Once entered, data can be converted into a variety of popular two- and three-dimensional chart types. Charts then can be enhanced with color fills and blends and with scaleable ATM fonts. In addition, PM Chart is compatible with Micrografx' extensive ClipArt library which includes over 17,000 vector symbols. Micrografx maintains the largest symbol library in the PC market, and compatibility with that library should help make PM Chart an invaluable tool for most OS/2 2.0 customers. Users can, of course, purchase Micrografx Charisma or Designer to create their own high quality graphics and incorporate them into PM Chart documents.

As with Micrografx' other PM applications, PM Chart was developed in the Windows environment. First, Charisma was scaled down to PM Chart under Windows. Then PM Chart was ported to OS/2 PM using Mirrors.

Dylan Greiner, Software Engineer, explained the porting process. "The first thing I had to do was take the DLLs and port them with Mirrors. That meant rebuilding the make and link scripts for each of them to include the Mirrors libraries, and removing the Windows libraries. Then I put them into the Mirrors resource compiler and just let them go. They went without a hitch."

"Next, I had to bring the actual application over. At the time, I was maintaining separate versions for Windows and OS/2 PM, so I had to work with some of the names to keep them

Porting Windows device drivers with Oasis takes about two days

separate. That took the longest time, just figuring out and making sure there weren't any conflicts. Then I had to drop it into our MIRRC compiler and the def file converter. After I got the application into OS/2, it was rebuilt. The next thing I had to do was run the application, just to make sure that all the ported DLLs came up. Then I had to start looking for where all of the DOS dependencies were, and I did find a couple. When I found one, I went in and replaced it with an appropriate function of the host library, if there was one for it, or used something like a DOS3CALL."

Richard Merrick expanded on how Mirrors can provide host independence. "We have two different ways currently to provide host independence. We have a host API which we created before Microsoft put DOS3CALL into Windows. And now we're adding support for DOS3CALL. Either way, there's a notion of providing host independence, layering your application away from the interrupts that a lot of people call under DOS."

"One thing worth noting," Greiner emphasized, "is that here is an OS/2 2.0 PM application being created for worldwide distribution in a just a few months practically by one guy. And most of that time was spent paring the application down."

AN OASIS FOR DEVICE DRIVERS

The Oasis Toolkit provides two functions in one. It provides a porting layer for Windows device drivers, and it acts as a DDK for creating PM presentation drivers.

Micrografx' implementation of Oasis is very clever. Normally, PM presentation drivers interface directly to the GRE. They have the responsibility of hooking out an array of up to 256 pointers (to graphics handling routines). This array is called a dispatch table. The Oasis DLL sits between a ported Windows device driver (linked against the Mirrors and Oasis Import libraries) and the GRE."

A significant difference between Windows device drivers and PM presentation drivers is in escape code handling. Windows can support more than 60 printer escape codes while PM only provides 14. Further, only 10 of PM's escape codes have direct Windows counterparts. In order to accommodate the escape codes written into the ported Windows device drivers, Mirrors has to map any additional ones into one of the PM user definable ranges, then Oasis has to map them back again when they reach the ported driver. According to Wes Bell, Software Engineer and Oasis Project Lead, "Our experience has been that it is about a two day process to convert Windows device drivers with Oasis."

Wes Bell

Leveraging on the interface that the Oasis DLL provides to ported Windows device drivers, the Oasis Toolkit includes a DDK for developing native PM presentation drivers. Bell explained, "Someone can create a presentation driver with the Oasis DDK that provides all the functionality required to support virtually any raster banding printer. The Oasis PMPDK (Presentation Manager Presentation Driver Kit), or DDK, front ends all three of the exported calls that are required of all hardcopy presentation drivers (OS2_PM_DRV_ENABLE, OS2_PM_DRV_DEVMODE, and OS2_PM_DRV_DEVICENAMES). A call to export OS2_PM_DRV_ENABLE is funnelled through Oasis, where all of the device context management and raster image processing occurs. Oasis takes care of all the rendering issues, such as the mapping of different colors and bitmap translations. A call to export OS2_PM_DRV_DEVMODE is passed to the presentation driver as a call to the function pdkDevMode. Generic code in the Oasis PMPDK completely manages such issues as writing and reading the presentation driver configuration data to/from the OS2.INI file. A call to export OS2_PM_DRV_DEVICENAMES is handled completely by generic code in the Oasis PMPDK. This is done using strings,

from a stringtable in the resource file, that are set by the person building a presentation driver using the Oasis PMPDK."

"The time that is normally required to write a PM presentation driver is about a year or two. With this toolkit, you should be able to have something going within a week, and be finished in a month or two. This is because all of the rendering issues, all of the low level stuff, is handled by Oasis. When the presentation driver gets called to create a new device context instance, the driver is called from Oasis to report its configuration. After that, all of the rendering issues are handled inside of Oasis, and its raster image processor creates a bitmap. When it gets to the end of a band or the end of a page, one function call is made from Oasis to the presentation driver built using the Oasis PMPDK, instructing the driver to write the band. It gets the bitmap, and may compress or reformat it depending on the physical device to which the output is destined. Then it sends the output to its destination by making one or more calls to a single Oasis function that manages all issues related to the actual output destination. That is, the output from a driver is handled transparently by Oasis when the driver calls back to Oasis with the output data, regardless of whether that output is destined to a file, to the OS/2 print queue, or to a physical port (and, then, to a physical device)."

In the short term, Oasis provides an efficient means of building a comprehensive library of OS/2 PM presentation drivers. In the long term, Oasis will provide a central point of maintenance. Because drivers will be written to a standard Oasis interface and to their own devices, changes that affect all drivers can be made within the Oasis DLL itself. "Oasis is the only place where calls are resolved and where all the mapping occurs," Bell conveyed. "If there's a bug in some driver that may affect other drivers, one change to Oasis will fix the problem for all of them. The more drivers that are coded to Oasis, the more consistent it will be. All of the new presentation driver standards for OS/2 2.0 that are supposed to be implemented across all drivers actually are being implemented in Oasis first. It really will be a new standard for consistency of operation for drivers."

MORE QUESTIONS FOR THE DEVELOPERS

Mirrors Team

Developer: How are you handling product testing for Mirrors?

Richard Merrick, Director of Systems Development: Testing Mirrors is like testing Windows. In fact, that's the approach. Test all the Windows functionality, and you're testing Mirrors. It's a big job, and its one we are discovering that Microsoft evidently hasn't even done quite so thoroughly.

Michael Harrison, Software Engineer: One of the test applications lets you create any kind of window that is legal under Windows, specifying all of the parameters for the CreateWindow call and moving it around on the screen, making sure that it acts the same way under Mirrors that it does under Windows.

Merrick: We've discovered a number of Windows bugs along the way, really sort of strange and unusual things. But you've got to know the bugs in Windows to know whether or not (your problem) is a Mirrors bug.

Derrel Blain, Technical Writer: That's another condition of the documentation, to be able to explain the differences between how things should work and the way they actually work, with examples.

Merrick: It's sort of a finer resolution definition of what Windows is, rather than what it should be. That's important in terms of supporting real applications.

Developer: Does Mirrors support DDE?
Jeff English, Software Engineer: Mirrors supports Dynamic Data Exchange both from Mirrors application to Mirrors application and from Mirrors application to native PM application. And there's a translation that goes on there because Windows and PM handle DDE a little bit differently. Same messages, but how they handle parameters is a little bit different.

Developer: How do you handle the Multiple Document Interface for Windows ported applications?

English: The Multiple Document Interface is implemented in Mirrors to function the same way as under Windows. In PM, there is no native Multiple Document Interface support; you basically have to make your own. In Windows there's an API for it, but it is really nothing more than a set of messages used to handle multiple documents within the same application.

PM GRE Team

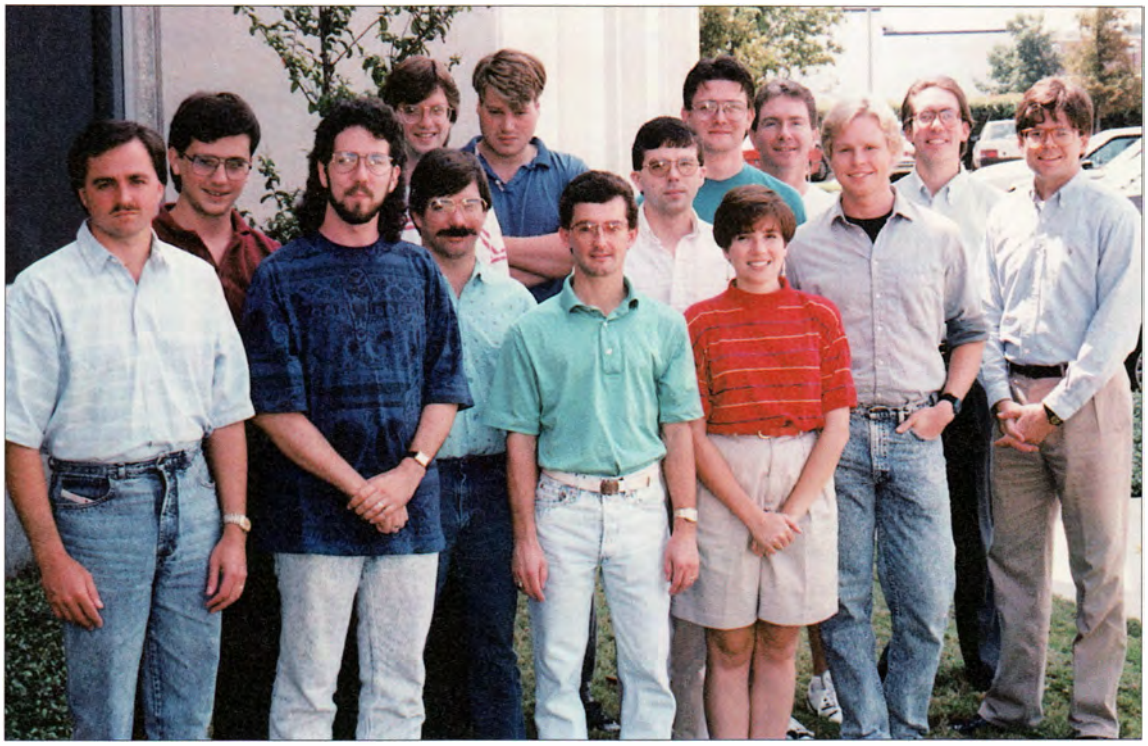
Developer: What is your view of Windows and OS/2 performance and directions?

Dave Raymer, Software Engineer: Anybody starting applications now from scratch needs to take a long, hard look at using OS/2 over Windows. Currently Windows is a shell, and that's all you can say about it. You hear all this great talk about Win32, and about beta testing, but it is all vaporware at this time, we haven't seen anything. When OS/2 2.0 is released shortly, they're going to have a 32-bit GUI environment that does true preemptive timeslicing. It's going to provide everything that Microsoft says Win32 is going to do, and more. If they're not already in Windows, then the learning curve is going to be the same, if not lower than the learning curve to go to Windows. PM is a lot more intuitive than Windows.

"If you take a Windows application and you take a PM application using native PM calls, PM is still noticeably faster than straight, basic Windows. And I really think with 2.0 we're just really gonna blow the doors off the graphical user-interface world."

Merrick: OS/2 has a consistent API across all subsystems, instead of dealing with the Windows APIs for some functions and with interrupts for others.

Developer: What kind of performance should we expect from the new GRE?



Development Team L-R: Kurt Delimon, Derrel Blain, Michael Harrison, Steven Urquart, David Consolver, Charles Forsythe, Jeff English, Dave Raymer, Dylan Greiner, Ed Qualls, Liz Bode, Roger Hagen, Wes Bell and Scott Lawson.

Ed Qualls, Software Engineer and GRE Project Lead: My feeling about the kind of increase we'll see with native 32-bit PM apps (using the native 32-bit GRE in a fully 32-bit OS/2) is that a twenty percent increase in speed not only is meetable, but beatable. If it isn't considerably more, a full third increase, I will be surprised. Any PM app that had originally been written for 1.x, recompiled and relinked will see that kind of increase.

In any applications where there are shorts, or 16-bit values being used, those will slow down a 386 or 486. The application developers will have to watch out for the 16-bit based optimizations they used. If they're in 32-bit mode and they start accessing data on 16-bit boundaries, they will be interfering with the data prefetch and that will cause them not to see as much increase in speed as they could.

Raymer: Especially on a 486. With the instruction prefetch and the data prefetch of 128-bit blocks, all of a sudden they could find themselves off by two or more bytes and missing half a long or missing half of a data structure. Then the CPU has to fetch another 128 bits, and that eats clock cycles. If you get off by a few bytes somewhere, its going to filter down through the rest of your code.

EXECUTIVE OUTLOOK FROM GEORGE AND PAUL GRAYSON

Developer: Can you discuss your relationship with IBM?

George Grayson (President and Chief Operating Officer): We made a lot of effort in the early days of OS/2 to get involved, and to help play a similar role for OS/2 to the one that we've played with Microsoft and Windows. We had a role with Microsoft in helping Windows become successful, and helping Microsoft understand the market for Windows. A lot of the things that we're trying to bring to the table are based on what we learned from having worked with Microsoft. I think that in many respects Paul and I are realizing a dream in being able to provide the work we're doing for IBM right now.

Developer: Has the investment been worth it?

George: If I were to base return on investment on the sales of OS/2 applications, that would not be true. Do I think that it's going to be worth it? Absolutely. I expect it will pay off very handsomely. I think there's going to be a significant market for OS/2 and for OS/2 applications. I believe the market requires a better operating system than MS-DOS, and that OS/2 architecturally represents that operating system.

Developer: Who are your primary customers?

George Grayson

George: The primary customers for us have been Fortune 1000. We've organized our company in that way. We are starting to try to change that focus, looking at being more broad-based. We have a direct sales force that we spend more money on than we spend on direct advertising. We've recognized, especially in the last year or so, that there's a broader market out there too, and we're trying to do a better job at addressing those through direct advertising. That is not because of our OS/2 work, that's because of recognizing that we weren't participating in that larger market to the extent that we thought we should be able to. OS/2 is going to let us do a better job at addressing that Fortune 1000 market, as well.

Developer: What are your key distribution channels?

George: We go through the main distribution channels and several direct dealers, software only dealers, and we have a corporate sales force. But the corporate sales force doesn't really take many orders, they get us on the standards lists, and they work inside the corporation to broaden the appeal of our products. But ultimately, we tend to route the business through the dealer channel.

We're realizing a dream in the work we're doing for IBM right now

About forty-eight percent of our sales are outside of the United States. We started making a significant investment in the international market a number of years ago, back in about 1986.

Developer: What is your current view of the Windows and OS/2 markets?

George: Almost all of our revenue is from Windows applications. And Windows is, from our perspective, a very fine operating environment. It has its deficiencies. We hope to address those deficiencies in what we view as an evolutionary fashion, which explains Mirrors and our relationship with OS/2.

Paul Grayson (Chairman and Chief Executive Officer): We couldn't afford to work on OS/2 if it weren't for the success of Windows.

George: If we do this right, we're going to take all this Windows development and turn it into OS/2 development. You're going to use Windows to leverage OS/2 in the future. A lot of people feel that way. I'm convinced.

Paul: We're primarily selling applications, and we're selling Windows applications. As Windows has become more popular, we've focused less on the fact that they're Windows and more on the fact that they're the best applications in the market. The trick for us is how to move these products from Windows to OS/2 and still continue to focus on selling the applications and delivering better applications to our customers, and to minimize the incremental costs of doing so. One of the big issues for us, and of course all the other developers, is we have limited development resources, so we can't really put big teams on both Windows and OS/2 platforms, and that's why we've chosen the approach we have. To figure out how we can invest in our applications and maintain that investment through a migration in operating systems.

Paul Grayson

George: It doesn't matter how big the company is, you still have limited development resources. Look at Silicon Valley today. Programmers are being traded back and forth as fast as they can move. They're not adding new programmers nearly fast enough to do all the projects that need to be done in the U.S. or worldwide. So, the bottom line is we've got to be more efficient in how we develop software. Certainly in the future that may mean object oriented programming and stuff like that. But today it also can mean taking a tool like Mirrors, preserving all of your investment in Windows applications development (which was done by all of your top developers), and leveraging that into a strategy which yields what we believe are going to be some of the finest-performing OS/2 applications ever written.

Developer: What advice can you offer a startup company?

George: We used Windows as a technology lever, to leverage ourselves into the business, to get a jump on all of our competitors. I think clearly that a startup can do the same thing with OS/2. But they need to realize that they're not going to go from 0 to 50 million in a year or two. They're going to go through some really tough times. They're going to be a startup. They're going to have to behave like a startup. And they're going to have to do multiple revs of their application, and they're not going to be perfect the first time, they're going to have to deal with all of the problems of the operating system. But if they do that, if they stick to it, year after year after year, one day they're going to find themselves ahead of everybody else.

Paul: Anytime there's a new environment like a Windows or an OS/2, there are a lot of available niches where you can be a unique company. You don't have to be a "me too" company. And there's a danger that if you try to get involved in a mature environment like DOS is now, or even Windows, that you try to join the bandwagon too late. To start a company today to do Windows development, well that's a pretty "me too" idea. Everybody's doing that.

George: More people fail right now under Windows than succeed by a long shot, about five to one.

Paul: Every software vendor will say on the cover of PC Week "we're doing Windows development", so that's not unique anymore. Doing OS/2 can still be unique, particularly if you pick a niche that really takes advantage of OS/2. Create some new application, or do something that nobody has done under DOS or Windows, or not yet under OS/2. You can get the kind of attention and free publicity, a lot of help from IBM, and things like that that I think can help get a company out of the garage.

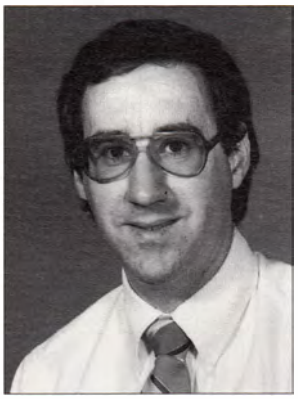
George: And you're in a small but rapidly growing market if there's any success at all in OS/2. And what you want to do as a new company is get into a growth market, not to come into a mature market and try to fight out some established player for market share. Really I think the startup opportunities are around new environments.

Manufacturing

Distributed Automation Edition: A CIM Enabler Platform

Pat DeNardo

Lynne Derrick



Grayson Randall

by Pat DeNardo, Lynne Derrick and Grayson Randall

IBM CIM ARCHITECTURE

Computer Integrated Manufacturing (CIM) architecture is designed to support application development in a highly distributed environment. The CIM architecture recognizes three basic building blocks that are common to all applications. These blocks provide the ability to communicate; the ability to manage data; and the ability to provide and accept data to and from both people and devices.

The IBM CIM architecture defines a layered set of services to allow sharing of data across an enterprise. The bottom layer consists of the operating system and hardware; the top layer is the application function; and the middle layer, common services, is supplied by the application and system enablers.

The application enabler layer is defined as a set of common services that are consistent within a group, such as tool control. Not all CIM applications provide tool control, but all CIM applications that do provide tool control provide the same basic services. The system enabler is defined as a set of common services that are common across all applications. Primarily these services include support of communications, data management, device management, and presentation services. (See Figure 1.)

With use of the personal computer becoming widespread throughout the manufacturing environment, Operating System/2® (OS/2®) has become IBM's manufacturing operating system on the plant floor. The CIM architecture, however, requires solutions that can run on IBM's various computer platforms, such as VM, MVS, and OS/400®.

The CIM architecture extends past SAA™ to encompass international manufacturing standards such as MAP, defacto standards such as TCP/IP and AIX®, as well as specialized manufacturing controllers such as the ARTIC cards. In order to satisfy this requirement for a system enabler in the IBM CIM product line and also to support non-IBM developers in adopting the IBM CIM architecture, IBM announced a collection of products called Plant Floor Series™ Distributed Automation Edition™. (See Figure 2.)

To increase the productivity of programmers, it is necessary to have common, portable techniques for communicating in a distributed manufacturing environment. The ability to exchange information in a network without programmer knowledge of individual protocols and data representations is a very important requirement for network users. This article addresses the concept of common, portable interfaces that can be used in a highly distributed and heterogeneous network environments.

Distributed Automation Edition currently runs on IBM PS/2® and hardened Industrial Computer Systems running the OS/2 operating system, and on System/370™, System/9000™, and Enterprise System/9370™ systems running the Virtual Machine (VM) operating system. A subset of the Distributed Automation Edition Application Programming Interfaces (APIs) and functions is provided for the Series/400 systems running

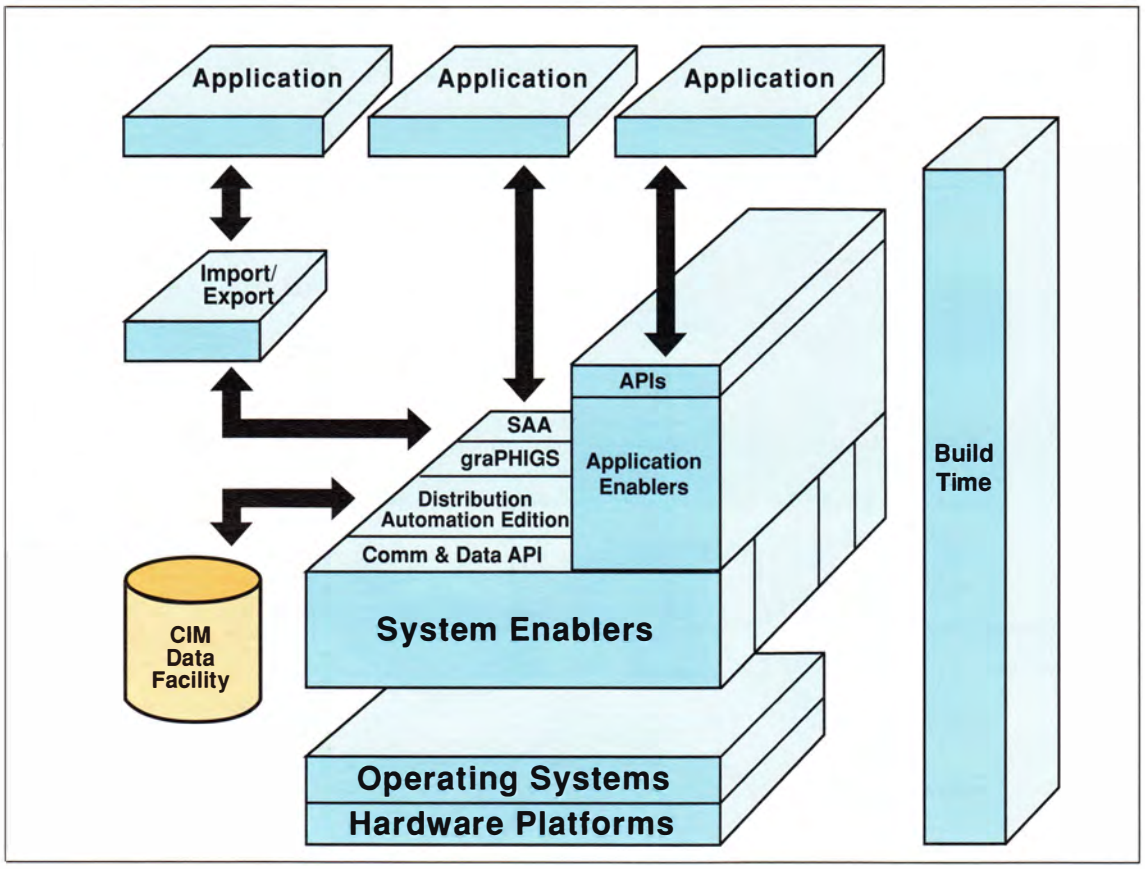


Figure 1. IBM CIM Architecture

OS/2 has become IBM's manufacturing operating system

the OS/400 operating system, for the RISC System/6000™ running the AIX operating system, and for PS/2's running the DOS operating system. The Distributed Automation Edition subset provides 'client' capability, allowing applications to request services from a full Distributed Automation Edition system running either OS/2 or VM. The application need not be programmed differently because the actual services are performed remotely on a different type of computer. The application running on the RISC System/6000 or AS/400®, in fact, could be recompiled and run on the OS/2 or VM system. Application portability is an important aspect of the architecture.

Communications with systems which do not run Distributed Automation Edition are handled by the Host Link products in the Distributed Automation Edition family. Services are provided to interchange transactions or files with IBM and non-IBM systems with built-in functions to help applications communicate with IBM's IMS, CICS, MAPICS and COPICS.

Distributed Automation Edition is a symmetrical client-server messaging system. That is, for any request, a computer running Distributed Automation Edition can be the source of the request, the supplier of the service (for example, the supplier of a database record), or both. Subset configurations called "Entry" products, which operate only as clients, are available at a lower cost. Communications among applications and functions are through queued messages.

DISTRIBUTED AUTOMATION EDITION FUNCTIONS

The functions provided by Distributed Automation Edition fall into four categories: communications, data, user interface, and device. (See Figure 3.)

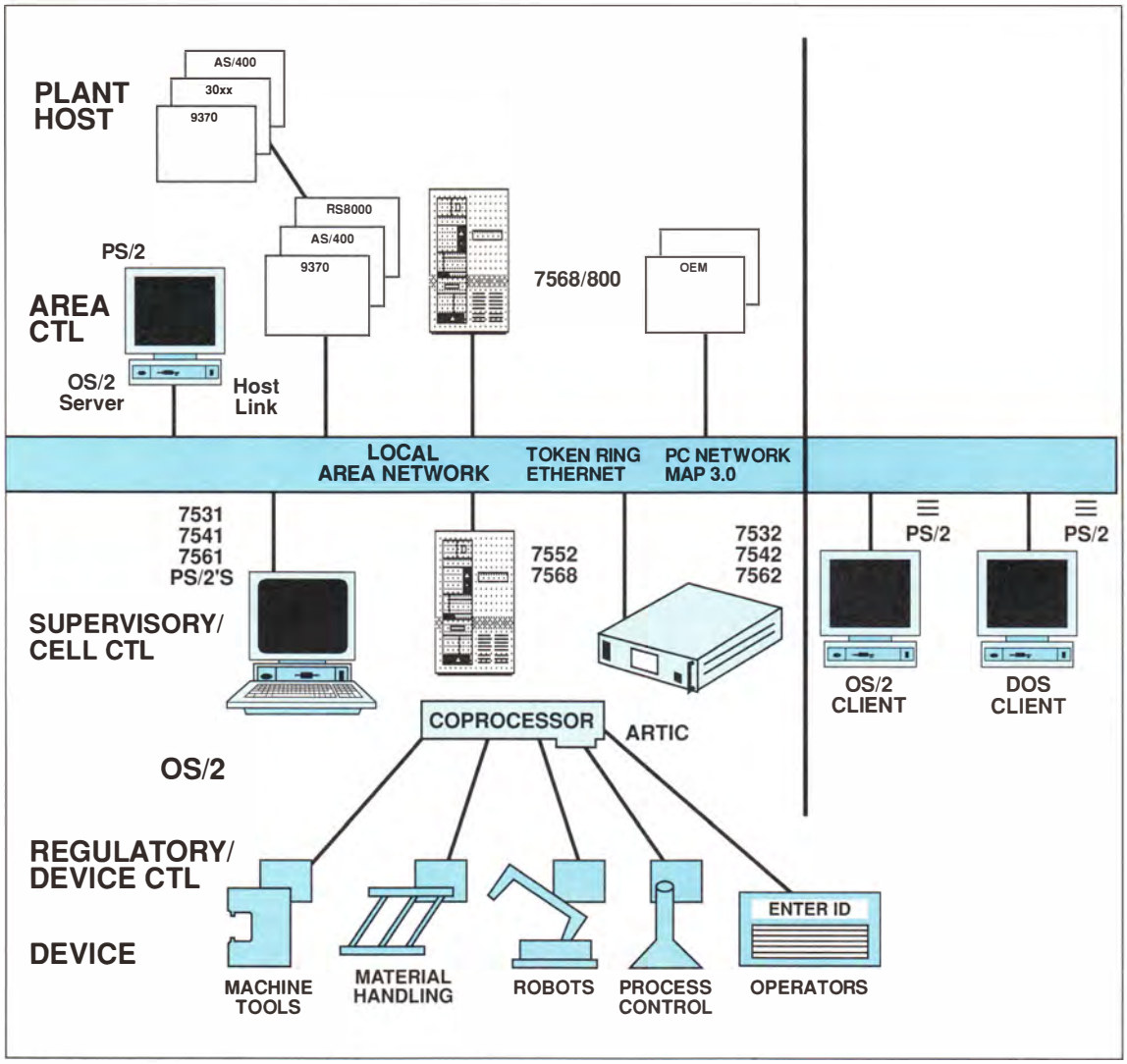


Figure 2. CIM: A 3-Tiered Manufacturing Universe

The Communications Component

Almost all applications need to communicate with other devices or applications in a network. Distributed Automation Edition supports a highly distributed network where resources (e.g., data, devices, applications) can reside on the same node as the application, or on any other node, independent of the platform and the operating system. The network protocols that are supported by the communications component are NetBIOS, APPC, TCP/IP, and MAP.

Distributed Automation Edition supplies the application developer with a set of APIs to do generic SENDs and RECEIVEs. Data is sent to a logical resource name using the SEND API. The physical location of that resource is bound by a local configuration table that the user manages by employing the configuration management utilities supplied with Distributed Automation Edition. This allows the application developer to write an application using logical names, and bind those logical names to physical resources at run time. Because the logical name mapping is done at run time, it is possible to change the physical resources dynamically while the applications are running.

The user employs the configuration utilities to define the physical locations of logical names at build time. These definitions include items such as destination operating system, network to be used, and network protocol to be used. This concept of configuring resources and control blocks allows Distributed Automation

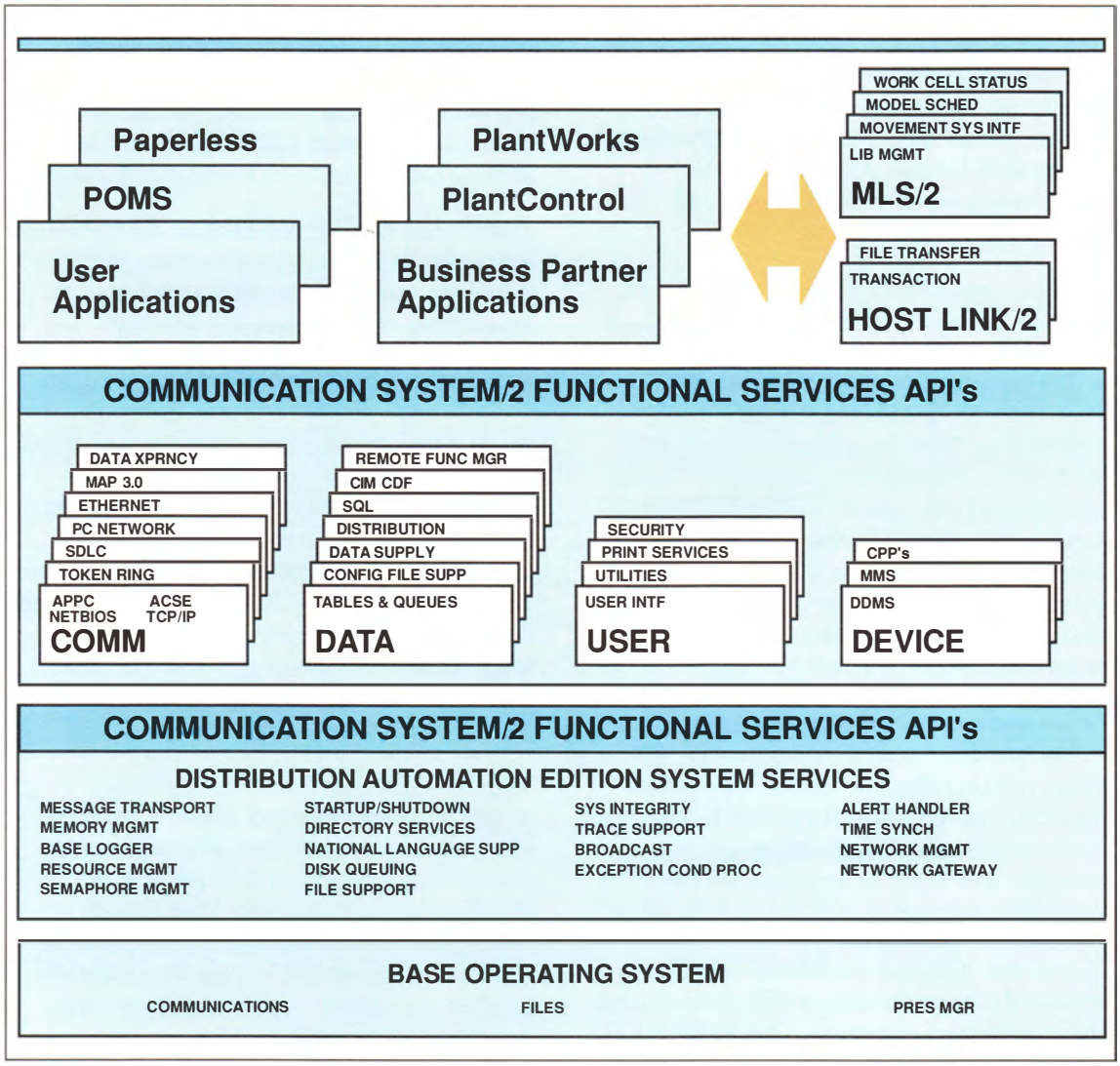


Figure 3. Distributed Automation Edition Interfaces

Edition to provide transparency to the application.

This allows a CIM application to be written using only logical names. The operational system then is configured to correspond to a particular installation configuration. If there is a failure in the system, it is possible to reconfigure dynamically and to redefine one or more of the physical resources. In addition, if the user wishes to use the application in another location that has a different network installed(e.g., TCP/IP or MAP), then all that is required is to reconfigure the Distributed Automation Edition node. No application changes are required.

The Distributed Automation Edition system enabler is queue driven. All messages that are sent are placed on queues defined for each resource. All resources then read messages from their queues for appropriate processing. Each message has a header that defines the sending resource, the priority of the message, special flags and the size of the data. This can be used to return responses to a request. It is possible to make these queues disk resident to assure that messages will not be lost during a node failure. Whether a queue is disk resident or memory resident is defined by a threshold which is set at configuration time, and by the individual message priority.

The Data Component

The next functional area supported by the Distributed Automation Edition system enabler is data management. The Distributed Automation Edition APIs support the application's use of tables, queues, and relational databases.

Tables are multi-level, indexed data files for storing user information. Table members are accessed by a primary key and, optionally, up to five secondary keys. Queues are single-level, indexed files for storing user information. They are accessed first-in, first-out (FIFO), or last-in first-out (LIFO). Tables and queues both consist of members that are fixed length, user-definable records. The maximum number of members in a table or queue is predefined by the user at build time. Members can be added, modified, or deleted by an application program.

Tables and queues have the capability to have a principal and/or an auxiliary image. Either image can be defined to be memory or disk resident. This provides the ability to maintain performance using a memory image, to maintain data integrity using a disk image, or to implement a variety of other combinations.

Distributed Automation Edition also allows relational databases such as SQL to be defined and managed as resources. This facilitates the use and integration of relational database products in a Distributed Automation Edition network. The relational database resource can be configured by the Distributed Automation Edition configuration utilities to support different relational database products on different platforms. Once configured, the application can code relational database requests without knowing on which platform the database will reside, or by which database product the database request will be handled. Distributed Automation Edition eliminates the need for database management system-specific precompilation processing and binding of the applications using relational database requests. It also manages all communications and data translations for requests among remote nodes.

An additional feature, not found in most database products, is the ability to make asynchronous requests. This gives an application the ability to start several relational database requests in parallel, and then to process the responses as the requests complete.

Distributed Automation Edition also provides a transaction management enabling facility for the creation of mission-critical On Line Transaction Program applications for manufacturing. (See the article *TXE: An Enabler For Client-Server OLTP* in this issue.)

As a system enabler, Distributed Automation Edition makes coding applications using relational databases very easy because of the single interface to all products. This, combined with a flexibility not found in other database products available today, makes very complex database applications easy to implement.

The User Interface Component

The IBM CIM architecture recognizes presentation services as a primary building block for a system enabler. For application writers, this always is a difficult part in developing an application. Distributed Automation Edition addresses this problem with a component called User Interface (UI). It supports applications by using a common interface for interacting with a user at a terminal. The UI component utilizes the client-server architecture and provides a distributed graphical interface. This allows location transparency, whereby a client application on one node can interact with operators on multiple remote nodes. Distributed Automation Edition now provides a UI WYSIWYG panel editor (an OS/2 application) and a compiler/linker which generate Presentation Manager screens. The panel description language (based on a tag language) and the screen APIs are platform-independent (which facilitates porting to other GUI environments).

The Device Component

The array of devices on the plant floor is characterized by a great variety of electrical connections, communications protocols, command sets, and data representations. The Manufacturing Automation Protocol (MAP) and Manufacturing Messaging Services (MMS) standardization effort approached this problem by attempting to induce device manufacturers to adopt a common interface standard. Distributed Automation Edition approaches the problem by providing a framework in which an application program can communicate with today's inconsistent devices using a consistent interface.

Distributed Automation Edition application program commands are modeled on the MMS standard and handle device-independent communications among user programs and devices. Distributed Automation Edition provides a device-independent interface for information received from, and sent to devices. This interface permits device types to be changed without redesigning and recoding the application program. This allows the application, for a given class of devices, to be written without knowledge of the specific device or its physical location. The device-independent command is converted to a device-specific command at run time, through the use of the configuration that was defined at build time. Not only does this make the coding easier, but the application now can be used in different locations where there may be different preferences of vendors. Simply by changing the build time configuration, the application will run with the new hardware.

SUMMARY

The IBM CIM architecture offers a technique for the layering of software functions. This layering allows for CIM implementations to consolidate commonly used functions out of the applications and into an enabler layer.

The Distributed Automation Edition product has some advantages that may not be clear from the architecture. Not only does Distributed Automation Edition increase programmer productivity as well as decrease software maintenance, but it also provides the capability to implement distributed applications where otherwise that would be impractical or even impossible. These abilities, to distribute applications, data, devices, and terminals throughout a Distributed Automation Edition network and to configure and manage those resources during both run time and build time, are advantages found only in a well layered system enabler that has been designed to be distributed.

Distributed Automation Edition meets the requirements defined by the IBM CIM architecture for a system enabler, as well as general requirements for any highly distributed network environment. The Distributed Automation Edition product should be considered as an operating environment for all networked applications. With Distributed Automation Edition, OS/2 now can play an important role in managing the manufacturing floor.

REFERENCES

IBM Corp., *Computer Integrated Manufacturing*, 1989, G320-9802-00

IBM Corp., *CIM Architecture*, 1989, G520-6686-00

IBM Corp., *Distributed Automation Edition Communications System, Technical Guide and Reference*, 1990, S33F-7722-02

IBM Corp., *Distributed Automation Edition Communications System, General Information Manual*, 1989, GC28-8049-03

Pat DeNardo, *IBM System's Technology Division, 1701 North Street, Endicott, NY 13760, (607) 755-6011. Mr. DeNardo is a Programmer Analyst in the NorthEast Region Application/ Development and Maintenance function. He joined IBM in 1977 and has a background in both hardware and software development. Recent projects include plant floor communications implementation and LAN communications for Distributed Automation Edition release 1.2.0.*

Lynne Derrick, *IBM System's Technology Division, 1701 North Street, Endicott, NY 13760, (607) 755-6537. Ms. Derrick is a Programmer Analyst in the NorthEast Region Plans and Controls function. She has been involved in projects associated with Endicott's Material Distribution Center and with the 9370 manufacturing line. Most recently, she worked on the development of Distributed Automation Edition release 1.2.0., and the communication of those functions to external customers.*

Grayson Randall, *IBM System's Technology Division, 1701 North Street, Endicott, NY 13760, (607) 755-5588. Mr. Randall is an Advisory Programmer in the NorthEast Region Application/Development and Maintenance function. He joined IBM in 1981 and has developed several projects associated with plant floor communications and Local Area Networks. Most recently, Mr. Randall had technical responsibility for the development of several new functions included with the 1.2.0 release of Distributed Automation Edition.*

Manufacturing

TxE: An Enabler For Client-Server OLTP

by Robert Orfali, Dan Harkey, and Robert Sujishi

WHAT IS TXE?

TxE, a short name for Transaction Enabler, provides a complete platform for creating Client-Server business solutions, including very demanding mission-critical On-Line-Transaction-Processing (OLTP) applications. TxE is a set of communications, database, screen-generation, screen-navigation, and transaction management tools which work together to allow developers to quickly develop LAN-based client-server applications.

TxE is part of the Distributed Automation Edition's Remote Function Manager services. It uses Distributed Automation Edition's platform-independent communications to provide remote services to clients (see DeNanardo listed in references below). The TxE server component is a transaction server that works with (and complements) the OS/2® EE Database Manager. The benefits of TxE include:

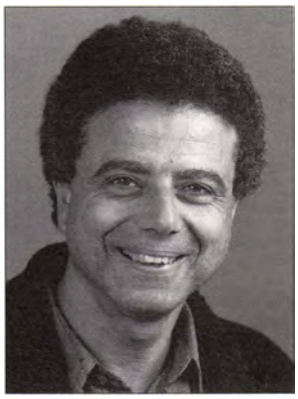
- **Fast Application Development:** TxE offers relief from endless coding and debugging and allows developers to focus on application-level design issues.
- **Less Programmer Training:** TxE hides the complexity of OS/2 multitasking, LAN protocols, database maintenance, and the tedious details of a Graphical User Interface (GUI). With a working knowledge of C and SQL developers can put together a LAN-based client-server application in less than three months.
- **Lower Cost of Ownership:** TxE offers a rich set of tools for managing servers in production environments.
- **High Availability:** TxE offers a *warm standby* server option for applications which require minimal downtime.
- **Lower System Cost:** TxE provides a robust OLTP platform on PCs. This translates into a low-cost hardware platform.
- **Better Applications:** TxE takes full advantage of GUI interfaces, resulting in more useable and intuitive applications.

THE ANATOMY OF A CLIENT-SERVER OLTP APPLICATION

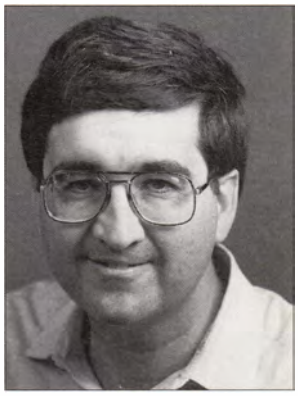
OLTP client-server systems are used to create applications in all walks of business. These applications include reservation systems, point-of-sale, tracking systems, inventory control, stockbroker workstations, and manufacturing shop floor control. These are, typically, mission-critical applications which require a 1-3 second response-time one hundred percent of the time.

The number of clients supported by an OLTP system may vary dramatically but the response time must be maintained. OLTP applications also require tight controls over the security and integrity of the database. The reliability and availability of the overall system must be very high. Data must be kept consistent and correct.

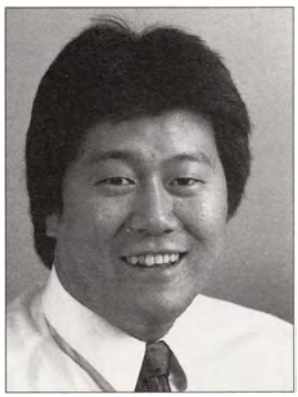
In OLTP systems, the client interacts with a Transaction Server instead of a Database Server (see Figure 1). This is necessary in



Robert Orfali



Dan Harkey



Robert Sujishi

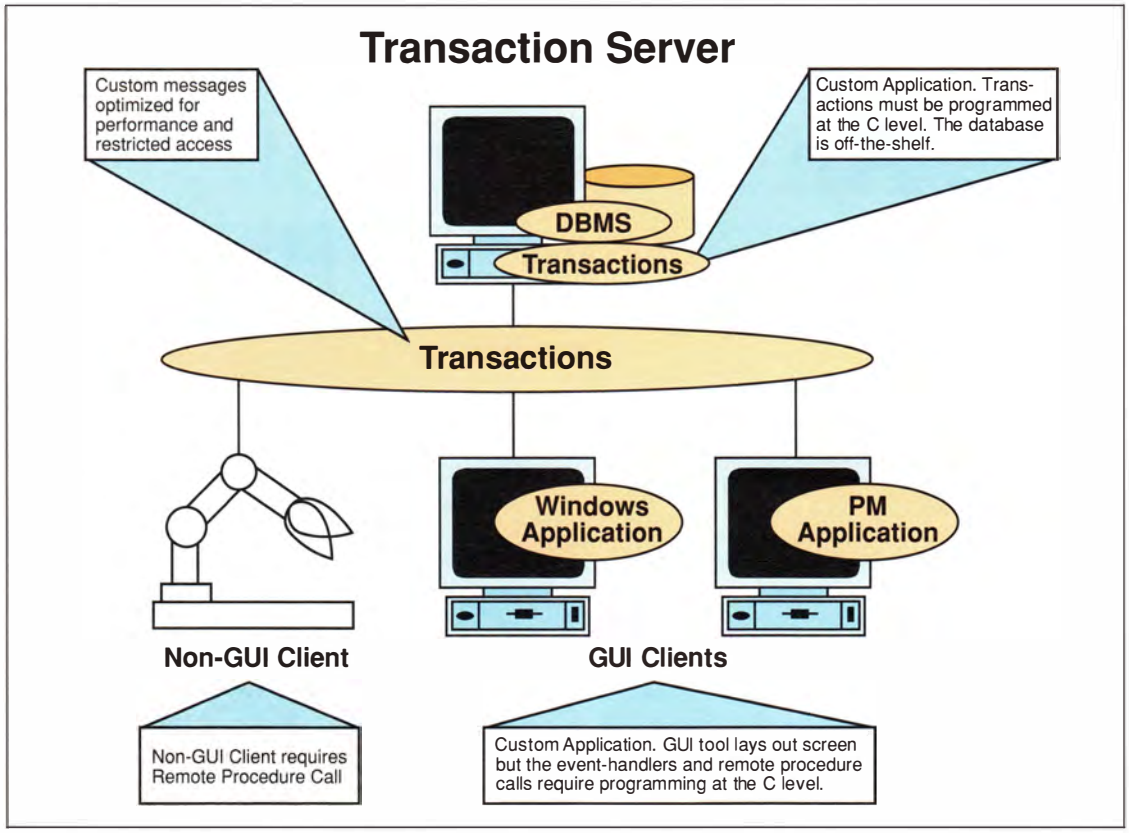


Figure 1. Transaction Servers for OLTP

The communication overhead in OLTP applications is kept to a minimum

order to provide the high performance these applications require. The OLTP client invokes remote procedures that reside on the transaction server. The server also contains an SQL database engine (such as OS/2 EE Database Manager). The remote procedures execute a group of SQL statements using an "all or nothing" transaction discipline against the server's database engine.

OLTP client-server applications require code to be written for both the client component and the server transactions (see Orfali, "Client-Server Programming ..." listed in references). The communication overhead in OLTP applications is kept to a minimum. The client interaction with the transaction server is typically limited to short, customized, structured exchanges. The exchange consists of a single request/reply (as opposed to multiple SQL statements). A peer-to-peer protocol is needed to issue the call to the remote procedure and obtain the results.

So developing OLTP in a client-server setting still requires intensive programmer involvement. Programmers are needed to write the transaction code on the server, work out the semantics of the network exchanges, and develop the client application. Non-GUI OLTP clients such as robots, testers, and ATMs may require less coding on the client side. But GUIs are becoming the predominant face of OLTP business computing. Rather than typing at the keyboard, users of GUI clients can interact with the OLTP server via their graphical icons, menus, and forms. To successfully develop OLTP client-server applications we will need creative new approaches to systems development and a new generation of tools (see Orfali, "Which Comes First…" listed in references). TxE is an example of this new generation of tools.

TXE: FROM RAPID-PROTOTYPE TO PRODUCTION

TxE helps in all phases of a client-server application's life-cycle:

1. In the early phases of an application TxE's rapid prototyping environment allows programmers to work with the user on the

application's front-end and create the graphical user interface quickly. The screens developed for the prototype can be used in the production version of the application.

2. Once the programmer has identified the application's main objects and functions, they use TxE to create the database on the server machine. Simple command scripts create SQL tables and indexes.

3. When the programmer is ready to code server transactions they can use TxE transaction templates and *make* files. TxE provides an environment for running the transactions and retrying them. It also provides an elaborate set of APIs for transaction services, and an optional software methodology.

4. Screens and the GUI can be developed in parallel with the server transactions. TxE provides event-triggers which couple the clients to the server. In OS/2 environments TxE takes care of creating threads for long-running client transactions. TxE also provides an environment and templates for screen navigation and an elaborate set of APIs for client services including an optional software methodology.

5. When the programmer is ready to unit test or system test, TxE provides the facilities for binding programs to the database, loading data into the database, and a run-time environment for integrating the clients and the server on the LAN. This includes elaborate database tools and traces which follow a transaction from its invocation through its execution.

6. During production TxE displays alerts and traces and allows fine-tuning and backing up of databases either manually or through triggers. TxE provides a transaction log and warm standby for high availability applications. This includes the tools for monitoring and reconciling a primary database with the secondary database or vise-versa.

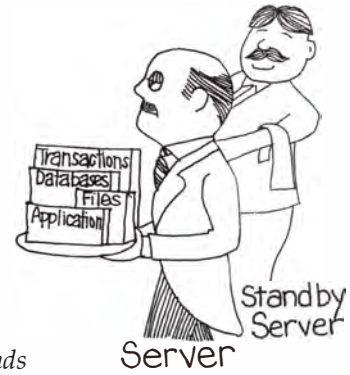
TXE: THE SYSTEM COMPONENTS

TxE architecturally is a Transaction Server. Clients and servers communicate through message-based remote procedure calls. The servers are shown in the top half of Figure 2 and the clients in the bottom half. The TxE system can be broken down into five distributed elements:

TxE Server: This is the element of TxE that executes the remote transaction against an OS/2 EE database. The server provides transaction services that hide the database structure and provide application-specific services.

Standby Server: This element is the optional *warm* standby server. It can act as a TxE server if the primary server fails. Standby is only needed for high-availability situations. The server can be configured to work with a journal on a separate disk using the special *roll-forward* capability TxE provides.

DBC: The Database Console element, or DBC as it is affectionately called, is used to create, configure, manage and maintain servers in production environments. DBC also serves as a remote front-end tool for TxE Servers (including standby servers).



Simple Clients: These are devices like *wands* or robots that have very simple presentation services. These clients do not require TxE's screen navigation services. However, they use the TxE Remote Procedure Call service.

GUI Clients: Both Presentation Manager™ (PM) and Windows® 3.0 (through third parties) GUIs can be used with TxE. A Dialog Manager-based solution is used to create low-cost OS/2 GUI client *front-end* screens that work with TxE servers. TxE provides API-driven Dynamic Link Library (DLL) functions for synchronizing screen navigation with TxE server transactions.

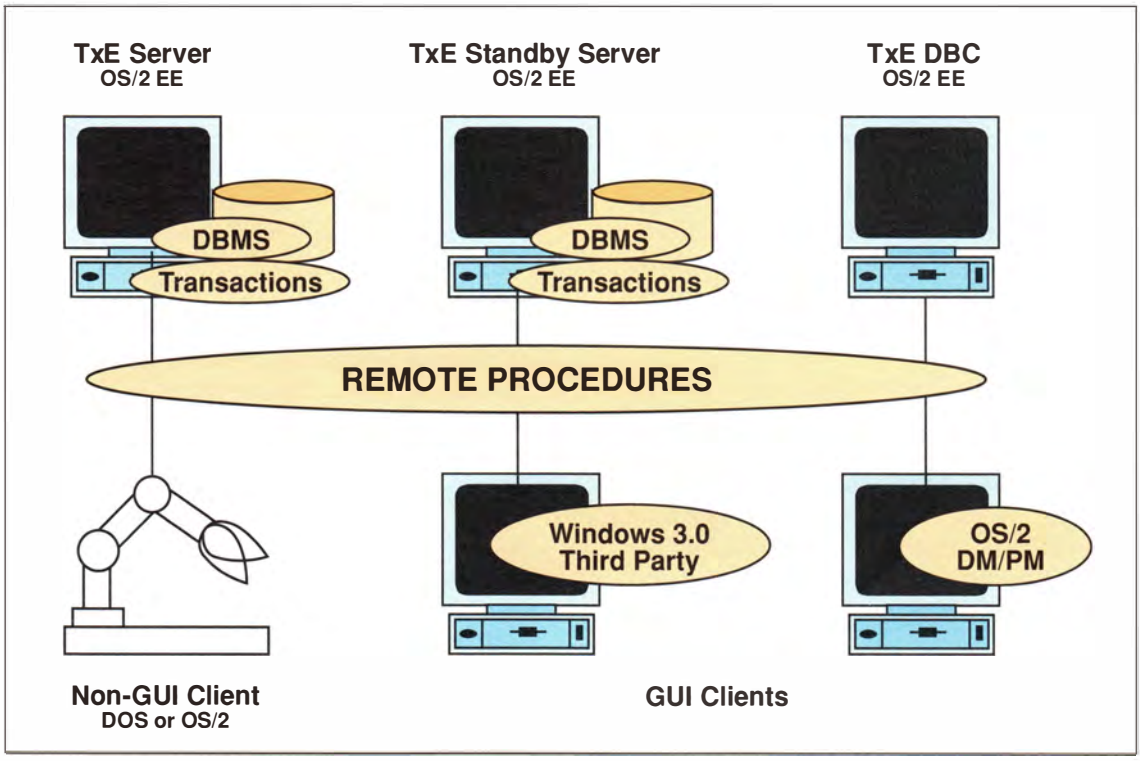


Figure 2. The Elements of a TxE System

WHAT MAKES TXE UNIQUE?

TxE was developed over several years to meet the needs of mission-critical manufacturing environments and factory floors. TxE provides a high-performance and reliable client-server platform on PCs. TxE servers were designed to replace the older supermini area-controllers as part of converting from a three-tiered architecture (hosts, superminis and PCs) to a two-tier architecture (hosts and PCs). The PC client-server platform was designed from the start to be as fast and reliable as the superminis it was replacing. In this section we discuss the unique features provided by the TxE platform for the creation of clients-server applications and for system management.

WHAT TXE PROVIDES TO HELP BUILD TRANSACTION SERVERS

The TxE Transaction Servers provide both the SQL database and the stored procedures which manipulate the data. The stored procedures are customer written DLLs which execute as transactions. TxE differentiates itself by providing the following unique server features on top of the OS/2 EE Database Manager:

- **Transaction Management Facility.** TxE manages a pool of transaction agents and a pool of incoming requests. The Transaction Management Facility matches an incoming transaction request with an available agent.
- **Transaction Priorities.** TxE supports two levels of execution priority for transactions: high and low. This means reports and other batch jobs can run in low priority while maintaining OLTP type response times for high priority transactions.
- **Remote Procedure Calls (RPC).** TxE provides a remote procedure call API which executes a simple message exchange between client and server and can move several Binary Large OBjects (BLOBs) as part of the RPC. Examples of BLOBs are image files, metafiles, database snapshots and voice. TxE's RPC also causes a customer-written transaction (a function in a DLL) to be loaded, passed its remote parameters, executed, and to have any results it produces returned.

- **Transaction Retry.** TxE will automatically retry a failed transaction and log the error. This retry capability is useful for recovering from deadlocks.
- **Transaction Rollforward Log.** TxE provides a transaction log facility which is maintained on a separate disk. If the primary disk crashes, all your transactions (since the last backup) can be re-applied to the database by TxE.
- **Warm Standby Server Capability.** The TxE platform provides a "warm" standby mode which dynamically redirects client transactions to the warm standby server so that when the primary server dies no data is lost. The downtime is less than five minutes (the time it takes to switch the standby server to primary mode).
- **Triggers.** TxE allows you to automate repetitive tasks such as backup with a time-based trigger mechanism.
- **Reusable Server Agents.** TxE maintains a pool of server agents that are persistently connected to the OS/2 EE database. A server agent can service multiple clients serially. This increases the OS/2 EE client limits without incurring the overhead of a START USE database. TxE manages all the database connections at all times.
- **Reusable Communication Sessions.** TxE's Remote Procedure Call is non-persistent. The communication session is relinquished at the end of the transaction. The server manages a pool of reusable sessions. This means that you can have more clients than the maximum number of sessions supported by your communications system.
- **Software Methodology (optional).** TxE provides a methodology for writing, documenting, tracing and navigating through transactions. Sample code, make files and documentation are also provided.
- **Multi-process Trace Facility.** TxE provides a trace facility which will display entries from different executing processes.
- **Status Console and Display.** The TxE Server console displays the executing transactions, statistics and alerts.

WHAT TXE PROVIDES TO HELP BUILD CLIENTS

TxE's Remote Procedure Call is powerful enough to be used in any situation where a client requests a remote service from a server. It is tailored to pass parameters (using ASCII strings for generality) to a remote function (DLL) that executes on the server. The actual parameters and remote functions are application-specific.

The Remote Procedure Call is designed to seamlessly integrate BLOBs into OLTP applications. It allows programmers to exploit the multimedia capabilities of workstations (such as image, voice, and animation) in client-server environments. For example, they can use the TxE support for large data objects to create BLOB repositories on the servers which work in conjunction with clients that capture, move and display large data objects. TxE's Remote Procedure Calls can be issued by clients from DOS or OS/2 environments.

The Remote Procedure Call is designed to seamlessly integrate BLOBs into OLTP applications

TxE also provides a set of client-enabling tools to help create GUI client front-ends for OLTP applications. Most current OLTP front-ends consist of a forms-like interface through which a user interacts with the server. The emphasis is on robust, high-performing, easy-to-maintain client environments. TxE provides a GUI version of this OLTP front-end environment using the OS/2 Dialog Manager. This is an integrated and fully supported environment for the creation of OLTP dialogs which includes the following set of client services:

- A front-end methodology for screen navigation.
- A WYSIWYG panel layout tool from a third party (see Pacheco article, page 81).
- Screen objects that are separate from the code, and thus easily maintained.
- Rapid-prototyping support.
- A library of PM User Exits. This includes support for metafiles, bitmap, fancy fonts, and image.

- Threads for long-running backend transactions (PM only).
- CUA Help facility.
- National Language Support for front-end services.

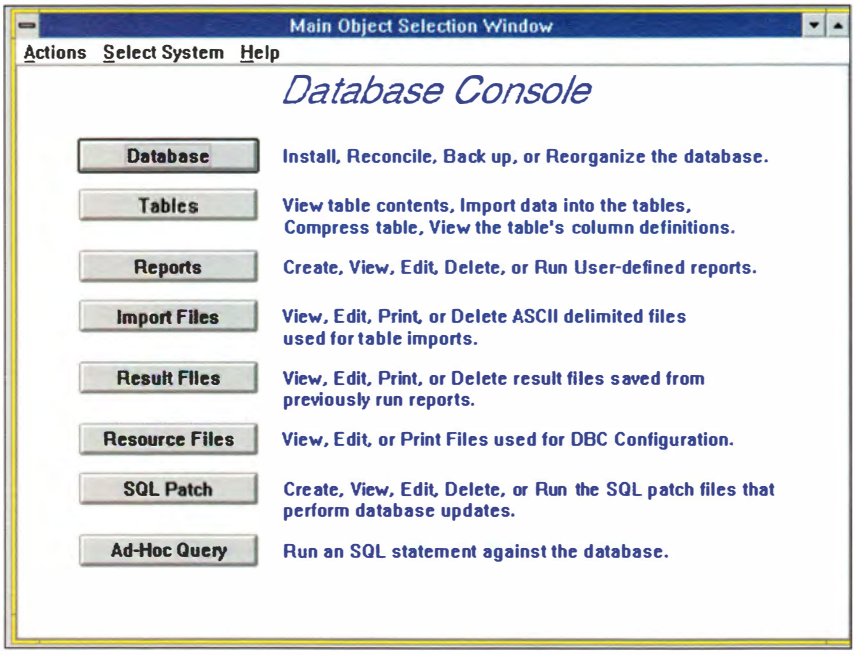


Figure 3. DBC Objects

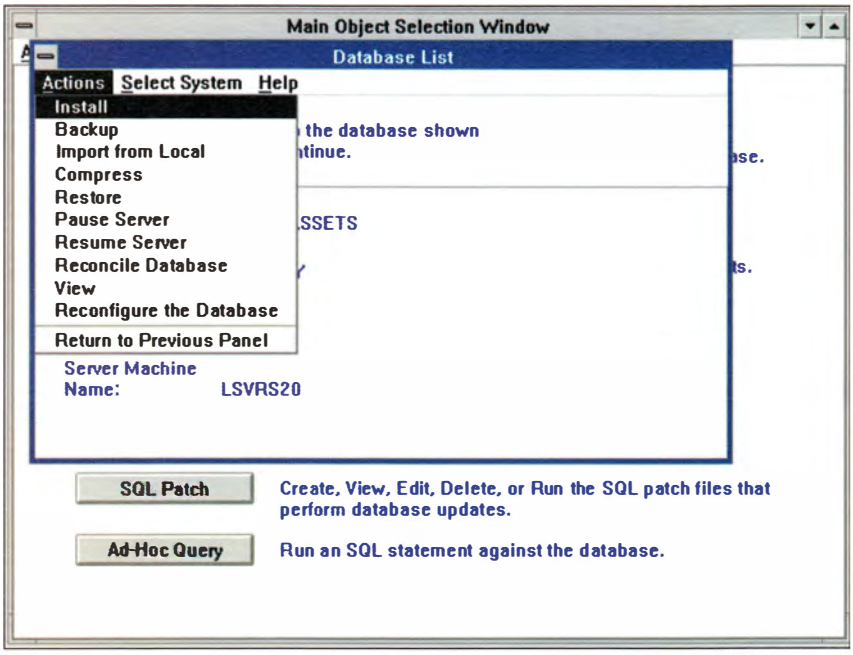


Figure 4. DBC Database Actions

Dialog Manager support for OS/2 is provided by IBM. A Windows 3.0 runtime is available from a third party (see Pacheco, ibid.).

WHAT TXE PROVIDES TO HELP MANAGE CLIENT-SERVER APPLICATIONS

Client-server OLTP offers the potential to downsize applications which traditionally run on superminis to lower cost PC servers and workstations. However these *downsized* applications don't come with a staff of database administrators, network experts and other MIS professionals. So, the million dollar question is how is a downsized system maintained in such an environment? How do we make sure the system won't perform miserably as the size of the database grows? How do we schedule preventive maintenance? Who will do it?

The answer is DBC. This is the excellent tool TxE provides for creating, configuring and managing servers in production environments. Some of DBC's functions are illustrated by the screens shown in Figure 3 and Figure 4. DBC provides database and system administration functions. It also provides dynamic SQL facilities for interactively executing SQL commands and running queries against the database. DBC provides three classes of functions:

- It serves as a database administration tool. It helps an operator or developer install and maintain an OS2/EE database and import data from different remote sources. It reconciles the server with a standby in case of failure. It also provides push-button facilities for fine-tuning the database. For example, at the push of a button you can cause DBC to reorganize and run statistics against every table in your database and then rebind all the programs.
- It serves as a reporting station allowing canned reports and ad hoc queries to be run dynamically against the OS/2 EE Database Manager.
- It allows an operator to run any SQL command against the database interactively.

SUMMARY

Client-Server OLTP applications require a new generation of tools that help create the client code, the server code, and the network transactions. The tools also have to support the application from the initial rapid prototyping, through build-time and finally through production. TxE is a tool that allows programmers to do all of this.

REFERENCES

DeNanardo, Pat, Lynne Derrick, and Grayson Randall. **Distributed Automation Edition: A CIM Enabler Platform**. *IBM Personal Systems Developer*, page 18, Fall 1991.

Orfali, Robert, and Dan Harkey. *Client Server Programming With OS/2 Extended Edition.* (New York: Van Nostrand Reinhold, 1991). IBM order Number: G325-0650.

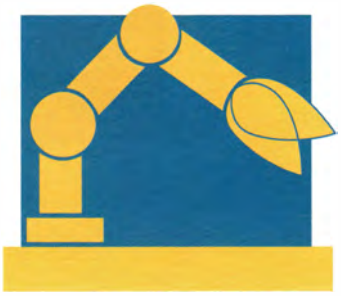
Orfali, Robert, and Dan Harkey. **Which Comes First: The Client or the Server.** *Database Programming and Design*, June 1991 and July 1991.

Pacheco, David C. **The Arcadia CUA Workbench: A GUI-Independent Tool for the Creation of Robust OLTP Front-ends,** page 81, *IBM Personal Systems Developer*, Fall 1991.

Robert Orfali, *IBM Corporation, 5600 Cottle Road, San Jose CA 95193. Mr. Orfali is an advisory programmer in IBM System Storage Products Division, Advanced Systems Development. He joined IBM in 1972 and is the architect of TxE and the co-author of* ***Client-Server Programming with OS/2 Extended Edition****. Mr. Orfali received an MSEE degree from the University of California at Berkeley.*

Dan Harkey, *IBM Corporation, 5600 Cottle Road, San Jose CA 95193, is an advisory programmer in IBM System Storage Products Division, Advanced Systems Development. Mr. Harkey is the lead developer of TxE, a client-server "Transaction Enabler" platform and is co-author of* ***Client-Server Programming with OS/2 Extended Edition****. He received an MSCS degree from Santa Clara University.*

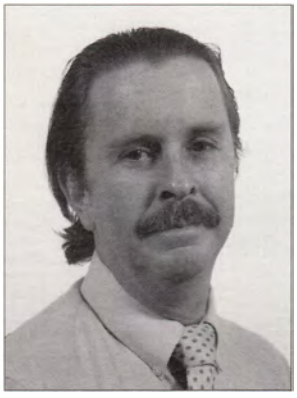
Bobby Sujishi, *IBM Corporation, 5600 Cottle Road, San Jose CA 95193, is an advisory programmer in IBM System Storage Products Division, Advanced Systems Development. Mr. Sujishi joined IBM in 1985 and is the team leader of TxE, a client-server "Transaction Enabler" platform. He received a computer science and mathematics BS degree from University of California at Davis.*



Manufacturing

PlantWorks: Application Automation Edition
The IBM CIM Application Enabler

William Belknap



Richard Hersh

PlantWorks is used to create and run manufacturing applications

by William Belknap and Richard Hersh

PlantWorks™ is a user-extendable system for creating and running manufacturing applications, providing an environment in which applications are driven by events as they happen. The four products that make up PlantWorks use the capabilities of the Operating System/2® Extended Edition (OS/2® EE) to provide session and process management, file and memory management, and I/O device support. Presentation Manager™ is used to provide the displays used to monitor the manufacturing operations. For communications between devices and applications and for access to distributed information, PlantWorks uses the system enabler (Distributed Automation Edition products), which also allows the installed PlantWorks applications to be moved as products, processors, and technologies change. (See Figure 1.)

As an application enabler, PlantWorks helps engineers create applications for the plant without the complexity of software programming. The system provides a highly graphical programming tool, in appearance much like flow-charting, that engineers can use for defining application logic. The application logic can be constructed from an extensive library of ready-to-use building block functions, and the library can be extended by programmers using the C language.

The engineer can also use fill-in-the-blank screens to define data, specify device configuration, assign application activities to various controllers, and specify operational alarms. The person creating the manufacturing applications also has access to supporting facilities for creating reports and colorful, animated graphical displays that depict the operation of the manufacturing application during run time.

Significant reductions in development and life-cycle costs are possible in this enabling environment because the people who understand the manufacturing process can create, modify, and maintain the application that provides the solution. Changes in the manufacturing operation, necessitated by market requirements, schedule changes, equipment replacement, or other causes can be made by the most knowledgeable people by modifying or enhancing the existing control applications. PlantWorks documents these modifications to support the needs of new or changing users. The cost in time and resources for reprogramming is significantly reduced, and skilled programmers are freed to create other software solutions.

PlantWorks takes advantage of IBM's OS/2 Presentation Manager™ to provide an interactive user environment that lets the user make selections with either a mouse or the keyboard; interactive windows provide the presentation support. A *point-and-click* approach plus user assistance features, such as HELP and menu driven selection, guide the user at both run time and build time. By using the interactive windows and menus, users can build completely self-documenting applications that include logic control flow charts, I/O diagrams, and alarm routes.

PlantWorks services, divided into run-time and build-time components, provide the means for creating and running applications. A collection of build-time definition services enable the creation, modification, and testing

of applications on the same networks where applications are running without disruption of those manufacturing operations. This means that changes to the manufacture of a product can be planned, created, and tested while a product is being made and then implemented without interrupting the manufacturing process. Essentially, build-time and run-time can occur at the same time, and manufacturing can proceed without downtime for installation of the application. (See Figure 2.)

PLANTWORKS BUILD-TIME SERVICES

Various PlantWorks build-time services increase user effectiveness by taking over the tedious chores of creating an application. Menus, interactive windows, panels with fill-in-the-blank fields, and graphically supported flowchart style tools increase the engineer's productivity. These services are part of the Definition Services/2 product, and include services for defining applications, data, I/O, chains, displays, alarms, reports, and security. There is a look-up service that supports selective searches for the other definitions. A few of the key components are described below:

Application Definition: With the Application Definition service, the engineer names the application and defines its overall attributes. This service allows engineers to partition applications to optimize future use and flexibility. The various definition tasks in creating an application can be divided among a group of users and these tasks can be performed in any order once the application has been defined using the application definition service. Many interactive applications can be concurrently operational, allowing the engineers to logically partition their process to simplify development and maintenance.

Extendable PlantWorks Function Library: PlantWorks provides a library of predefined functions useful in discrete, batch, and continuous manufacturing operations. These include math, database, logic, and chain types of functions among others. New functions can be created using C and then named and included in the library for use in creating or changing applications. This capability extends the applicability of the PlantWorks system to various manufacturing operations and provides opportunities to create functions and packaged groups of functions that are marketable to businesses with PlantWorks systems installed.

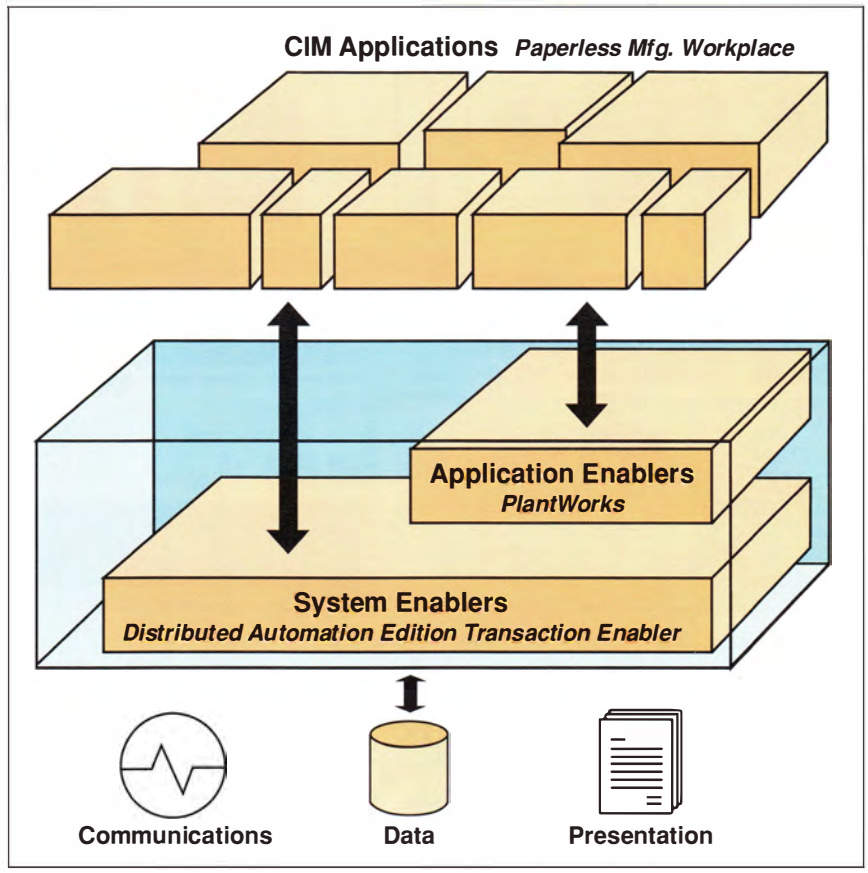


Figure 1. PlantWorks in IBM's Computer Integrated Manufacturing (CIM) Architecture

I/O Definition Service: Manufacturing operations can be handled by devices connected to nodes in the PlantWorks network in the plant. Devices like programmable controllers store data and communicate with nodes in the network by sending digital data over serial or parallel lines into PlantWorks via a co-processor card. The engineer defines the devices controlled by each node and the data read from and written to those devices.

The engineer determines how data is shared among devices and applications by defining I/O connections and points. I/O connections are combinations of physical devices such as nodes, Realtime Interface Co-Processors, communications lines, and devices like programmable controllers, that define the

device input and output data paths. I/O points are variables containing character, integer, floating point, or flag (Boolean) values, that store input or output data and reside on a device. The I/O points are defined by the engineer in terms of data characteristics, state changes, limits, and the alarms and chain logic to activate when the specified conditions are met. These points can be grouped into logical packets to simplify handling, and can be automatically polled at selected intervals and evaluated for change of state.

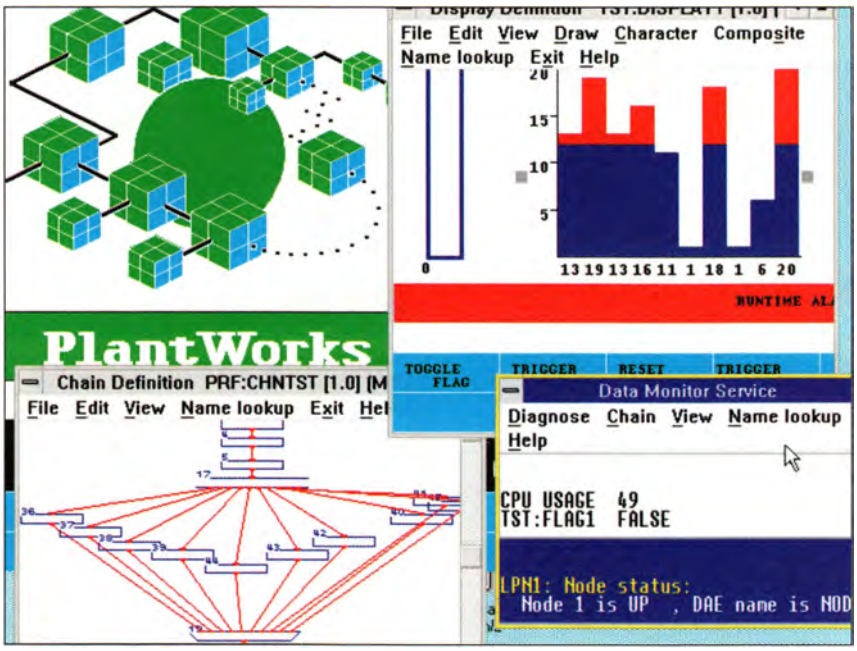


Figure 2. A Run-Time Display and Build-Time Definitions

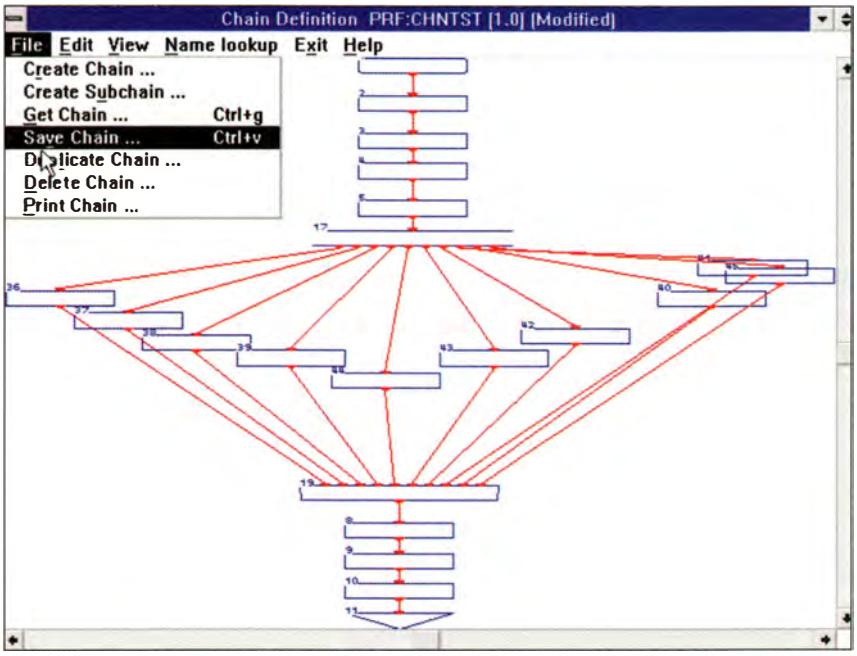


Figure 3. Chain Definition

Chain Definition: The Chain Definition service lets the engineer link a series of functions together in a chain in a graphical flowchart manner. This flowchart is a set of logically linked instructions that form the basis of the application, and is called a PlantWorks chain. For example, a chain might consist of functions that instruct a robot to pick up a part, insert the part, and tighten the part. To build the chain, the engineer selects predefined functions from a menu of configurable functions that support various application requirements. (See Figure 3.)

If a particular sequence of functions occurs frequently in chains, these functions can be linked together under one name as a PlantWorks subchain. Subchains can be used in chains exactly like functions, and allow the engineer to pass the application data and control information as parameters. This enables the engineer to modularize the common application logic into subroutines, increasing productivity through reuse.

Display Definition: This service is used to create interactive, graphical displays for the plant floor operator to use when monitoring manufacturing operations. These interactive displays are an integral part of the running application. Its advanced OS/2 Presentation Manager functionality enables the engineer to develop animated, graphical representations of the status on the plant floor directly, without the need for additional logic. For example, the engineer might create a display that shows the progress of a manufacturing process by using different symbols and colors to indicate which steps are being performed and their run-time status. (See Figure 4.)

The engineer creates the graphics, alarms, function key labels, operator messages, and data representations that appear on each display, as well as the logic that executes as the result of a function key selection. PlantWorks provides a graphics library of symbols to represent the plant floor devices which is

easily extendable or customizable by the engineer to fit the environment. The plant operator uses keyboard function keys that correspond to the display operator keys to control events, navigate through other displays, respond to alarms, and initiate other application activities. The result is a fully automated display that represents the activities in the plant and the options available to the plant operator to manage them. (See Figure 5.)

Alarm Definition: PlantWorks alarms notify plant operators about errors and important plant conditions as they occur, regardless of the action the application is performing or the particular display the plant operator is viewing at that moment. As an application is created, the engineer determines the conditions that will generate alarms, their priorities, the routes the alarms can take through the network, and the messages that will be associated with each alarm. When defining I/O points and chains, the engineer will define conditions that trigger alarms, then the alarm definition service can be used to define the alarms, the nodes on which the alarm message appears, how it looks, and the type of acknowledgement that is required for each alarm priority.

PLANTWORKS RUNTIME SERVICES

The run time services allow the plant operator to monitor and respond to manufacturing activities that are controlled by applications. The plant operator uses the run-time services to observe the progress of manufacturing operations, analyze data obtained by an application, respond to alarms, and generate reports. (See Figure 6.)

The run-time services are required on all nodes in the plant so that plant operators can access the services at any workstation. The plant operator uses the following run-time and utility services to control and monitor manufacturing activities:

Runtime Monitoring Service: The Runtime Monitor is used to view the dynamic, graphic displays that represent the state of a manufacturing activity controlled by an application. Plant operators can enter data on these displays and review data collected during manufacturing operations.

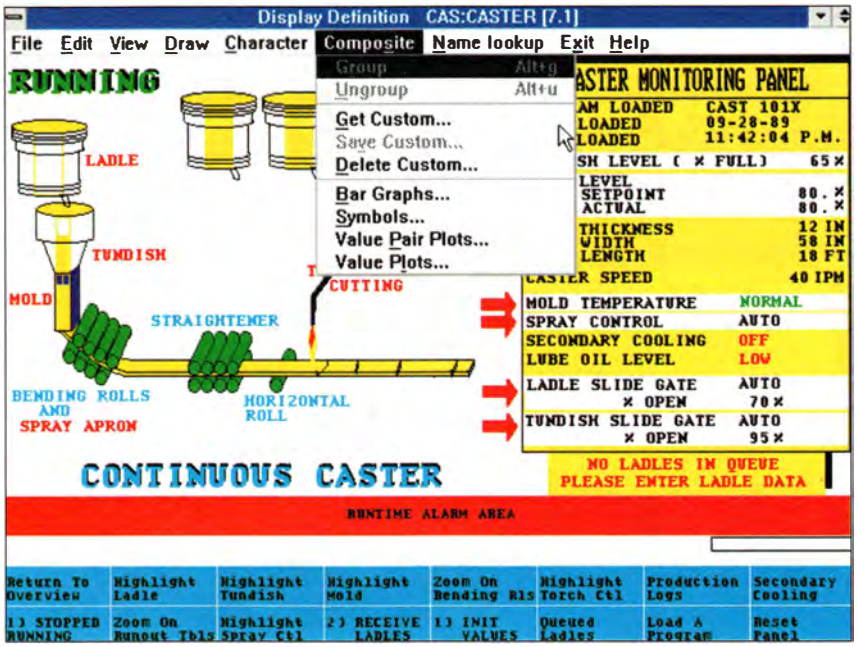


Figure 4. Interactive Display Definition

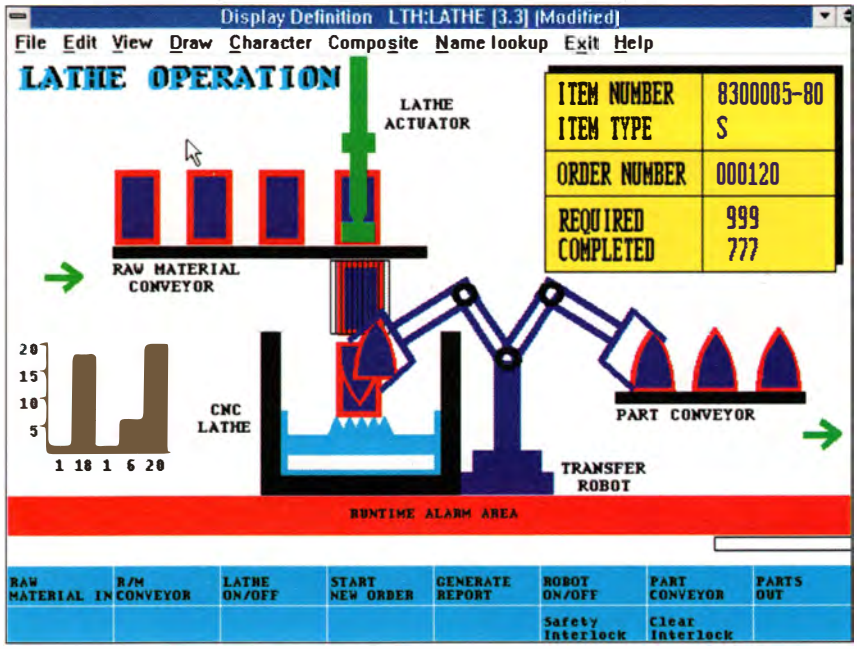


Figure 5. Runtime Display

Data Trending Service: The trend service is used to generate and view graphs that show data plotted over time for manufacturing operations that are currently being executed. Data is automatically updated as events occur. (See Figure 7.)

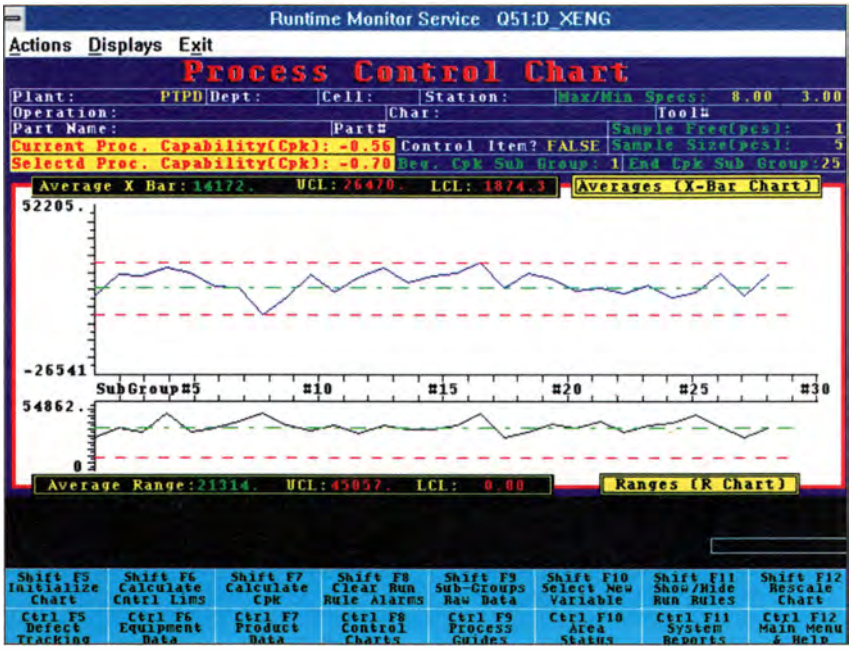


Figure 6. A Run Time Display of Plant Status

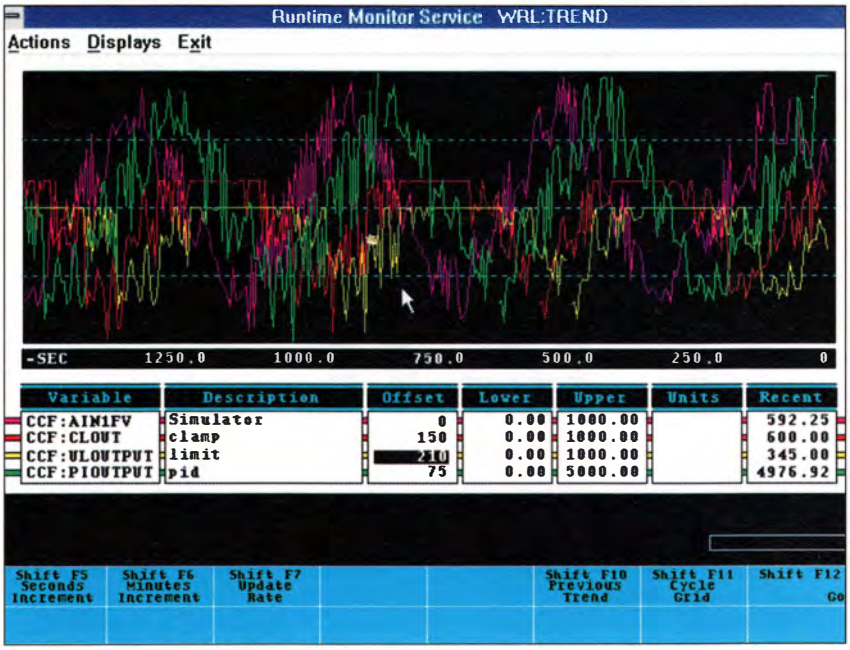


Figure 7. Trend Data Display

Alarm Management Service: When an alarm condition occurs during run time, the alarm message can appear in the alarm key areas of application displays or on the printouts generated by printers in the system. The alarm management service allows the operator to access, review, and acknowledge alarms routed to the plant operator's node and take corrective action as necessary. Alarms are automatically ordered by priority and routed to specific displays and printers according to the requirements established during build time. Alarms can be logged into the Alarm Audit Trail, which will have a record of the date and time of the alarm occurrence, as well as a description of the alarm.

Data History Service: A graphic service is used to review graphs of manufacturing data collected during a specific period of time. PlantWorks automatically records data based on criteria established by the engineer. From the recorded data, the plant operator can select the specific types of data, time interval, and time period to be graphed.

Report Services: The reporting service allows the plant operator to review reports of manufacturing data collected by PlantWorks. These reports can be triggered automatically from a PlantWorks chain, on a periodic basis, or manually by the plant floor operator or supervisor. The report is defined by the engineer during the application build process.

SUMMARY

PlantWorks provides the plant floor layer of the IBM CIM architecture that enables the creation and execution of manufacturing applications. It is a user-extendable system that provides a library of pre-defined functions, to which C language routines may be added, and services that make the capabilities of the operating system and system enabler layers transparent to the user.

PlantWorks increases the productivity of programmers and engineers. Using the highly graphical interface, engineers can employ the services of PlantWorks to create manufacturing applications without having to write a single line of code. Programmers, using C, can extend the functions available to those creating

applications. The completed applications are modifiable and testable on the same network where applications are controlling manufacturing operations. Entire applications and portions of applications are transportable across the manufacturing enterprise, so that an application created on one network can, with modification, be used on another network in the same plant or in another plant.

PlantWorks takes advantage of the highly distributed network environment of the Distributed Automation System system enabler and OS/2 EE to provide virtually unlimited manufacturing operations management through the build-time and run-time components.

REFERENCES

IBM Corporation and Measurex Corporation, **PlantWorks: General Information,** GC28-8250-01

IBM Corporation and Measurex Corporation, **PlantWorks: System Concepts,** SC28-8260.

IBM Corporation and Measurex Corporation, **PlantWorks: Planning and Implementation Guide,** SC28-8251-00.

IBM Corporation and Measurex Corporation, **PlantWorks: Execution Services/2 User's Guide,** SC28-8257-00

William Belknap, *IBM Corporation, 10411 Bubb Rd., Cupertino CA, 95014, joined IBM in 1982, developing Series/1 and PC tool control software in San Jose. He then worked with a process controls group, developing a Statistical Process Control package for the manufacturing floor. Since 1989, Mr. Belknap has worked with Manufacturing Systems Products at the PlantWorks Project Office in Cupertino. Mr. Belknap is an Advisory Programmer. He holds degrees in biology, computer engineering, and engineering systems from Stanford, U.C. San Diego, and San Jose State.*

Richard Hersh, *IBM Corporation, 1000 NW 51 St., Boca Raton FL, 33431, IBM Zip 4309, is a Staff Information Developer. He joined IBM in 1982 as an Information Developer in Boca Raton. He has worked as a writer, editor and planner on operating system and communication software publications, and as an instructor for IBM Technical Education. He currently works as an Information Development planner and coordinator on several software manufacturing projects, including the PlantWorks products. He received his BSJ from the University of Florida, and his MA from Florida Atlantic University.*

Manufacturing

PMW: The Paperless Manufacturing WorkPlace

Kate Ferriter

by Kate Ferriter

The PS/2® and Industrial Computer are used in many manufacturing environments and IBM is developing products on these platforms to satisfy our customer requirements. To meet the challenge of providing accurate and timely image-based information on the manufacturing floor, IBM has developed Paperless Manufacturing WorkPlace (PMW), an OS/2® application. PMW was part of IBM's October 1989 Computer Integrated Manufacturing (CIM) announcement and was shipped in September 1990.

IBM's CIM strategy focuses on the storage of shared information and its delivery throughout networks for manufacturing enterprises. CIM products are developed for SAA™ platforms with a system enabler layer added to assist the integration of data among multiple applications. As a key part of SAA, OS/2 provides functions required for many CIM products — SQL relational database, Presentation Manager™ and multitasking capabilities.

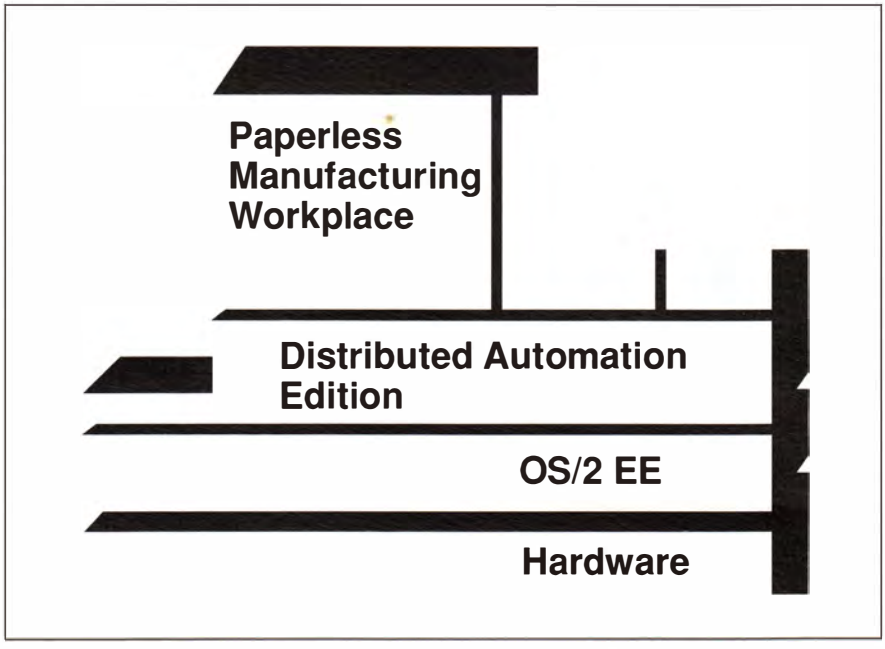


Figure 1. PMW Runs on Distributed Automation Edition and OS/2 EE

PMW uses Distributed Automation Edition™ common services as its system enabler (see article on Distributed Automation Edition in this issue). Distributed Automation Edition allows information to be shared among the *islands of automation* scattered throughout the manufacturing environment. (See Figure 1.)

To develop PMW, IBM worked with a small group of external customers to define their needs and determine what function should be in the product. Customers reviewed the product during all development phases. They critiqued PMW product specifications, and worked with prototypes and early code; providing over 100 Design Change Requests (DCR's) to the developer during product development. With their input, IBM has been able to provide a greatly improved product, with an end-user interface that is proving to be very effective in the manufacturing environment.

In today's manufacturing environment, machine setup, fabrication and assembly processes often are pursued with instructions and drawings that are complex and also may not reflect the latest revision levels for the product. These instructions, known as process plans or travelers, are attached to each part, component, or sub-assembly used in building a product. The process plan contains written instructions, drawings and pictures that are

used by the people performing the manufacturing operations on the shop floor. PMW applications allow the customer to control this information.

PMW APPLICATIONS

PMW is a series of modular applications for the creation, maintenance, distribution, and display of electronic (paperless) work instructions. The instructions can be used in all phases of the manufacturing process, and can include text, graphics, and/or images. PMW lets you start with a simple operator guidance application and evolve to more complex production operations that can be integrated with other IBM CIM solutions.

Paperless Manufacturing WorkPlace consists of the following application components (See Figure 2):

WorkServer

WorkServer is the central repository for PMW manufacturing data. At least one PMW WorkServer is required for a PMW LAN installation. The user of WorkServer is an engineer or information systems professional. Process plans are stored in the relational database and can be retrieved by WorkPlan

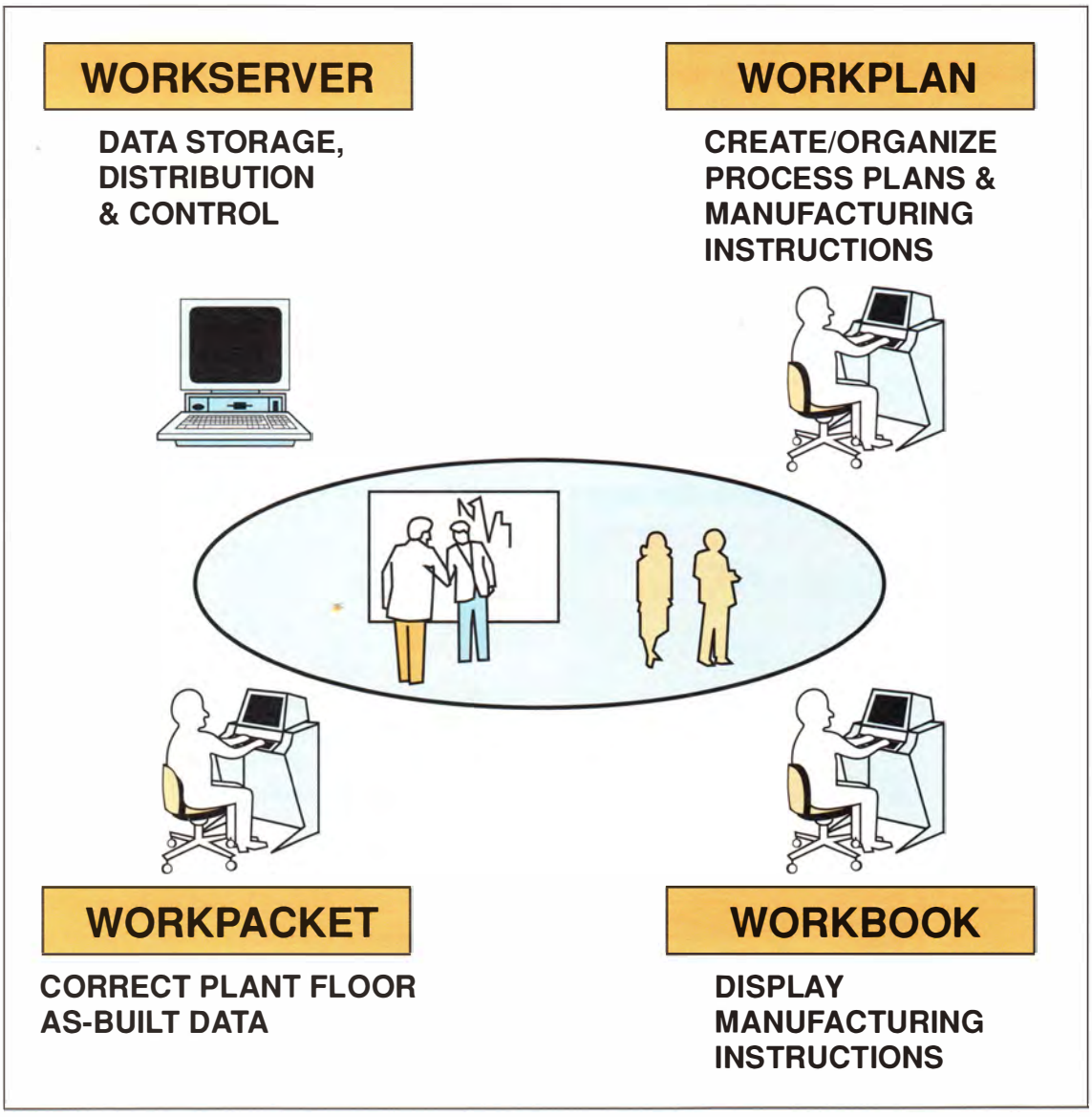


Figure 2. The Components of a Paperless Solution

and WorkBook. WorkServer is the keeper of the PMW objects, which are assembled and displayed in a multiple workstation system using a local area network. Customization capabilities allow tailoring to meet specific user needs.

PMW uses the OS/2 Database Manager's Structured Query Language (SQL). This allows a user to search for, and select objects being assembled based on parameters the user knows. For example, group technology can be used to search the relational database for objects which fit into a certain group technology family, like threaded fasteners.

WorkPlan

WorkPlan is the creation and maintenance application for the process plan or work instructions. It is the application that the manufacturing engineer or process planner uses to create the material that will be viewed later by manufacturing operators on the shop floor. It features transparent exits to editors like IBM CAD and OS/2 Image Support, native object editors used to modify the vector graphics and images used in developing or updating the process plan. The native editors used by PMW optionally can be exchanged with other Presentation Manager editors.

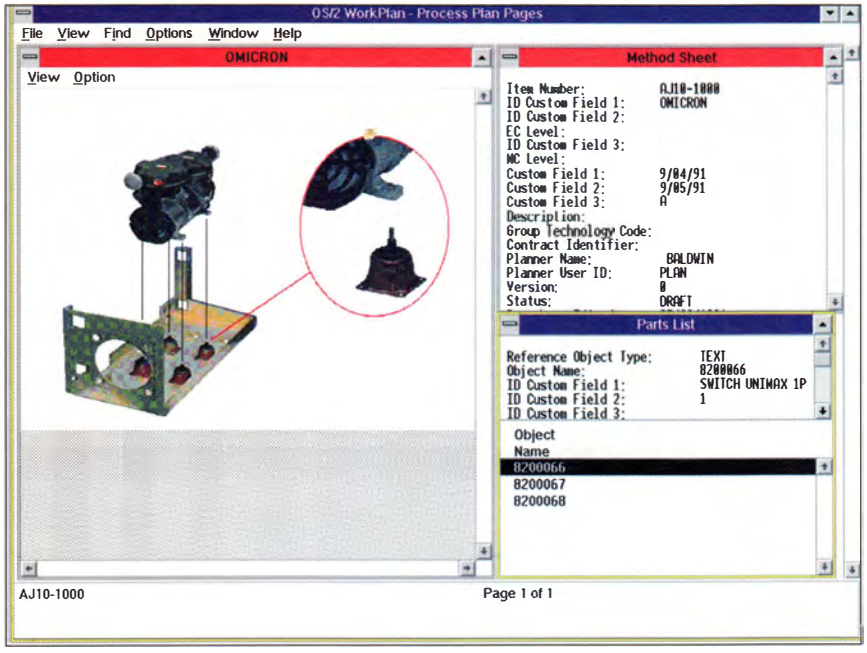


Figure 3. WorkPlan Can Assemble Multimedia Work Instructions

WorkPlan uses the WYSIWYG (What You See is What You Get) features of Presentation Manager to let the engineer locate and place objects in windows, then drag, place, and resize the windows on displays being defined for the shop floor operator. Use of the mouse is helpful for this user group. The WorkPlan user does not need to know any coding or write any tagged text in order to create these displays. The engineer also can incorporate user-written OS/2 programs resident at the workstationinto these displays. Such a PMW *custom program* is displayed in the same way as other object types, like pictures. When selected by the manufacturing operator, PMW shifts the focus to the other program, allows the operator to execute it, then returns control to PMW at completion. Each picture or text item integrated by PMW is an object which is stored in PMW's server. (See Figure 3.)

With WorkPlan you can:

- Add, change or delete an entire process plan
- Specify Screen layouts to be used by the shop floor operator
- Control the status of the process plans
- Integrate process plans with CAD graphics, references, scanned photos, video-captured images, textual and tabular data
- Use reference data shared among multiple process plans

WorkBook

WorkBook is the display application for the process plans or manufacturing instructions. It is the application that actually runs during the manufacturing process.

The user of WorkBook is the shop floor operator. WorkBook can be used with PF key input. This is by design, based on customer input concerning the computer experience of the shop floor user group.

WorkBook gives plant floor employees quick and dynamic access to the process plans, and always at the required revision level. Multiple versions of process plans may be worked simultaneously, depending on customer requirements. The operator can view and navigate through the following components of a process plan: Overview; Header; Operator guidance; Change history of the process plan; and Reference information.

The user can manipulate instruction displays easily, pan and zoom image and graphic displays, and scroll text. Process plans and operations can be queried, selected, and paged (see Figures 4 and 5).

The WorkBook workstation does not use the native object editors. PMW includes Presentation Manager show code which is included with WorkBook.

WorkPacket

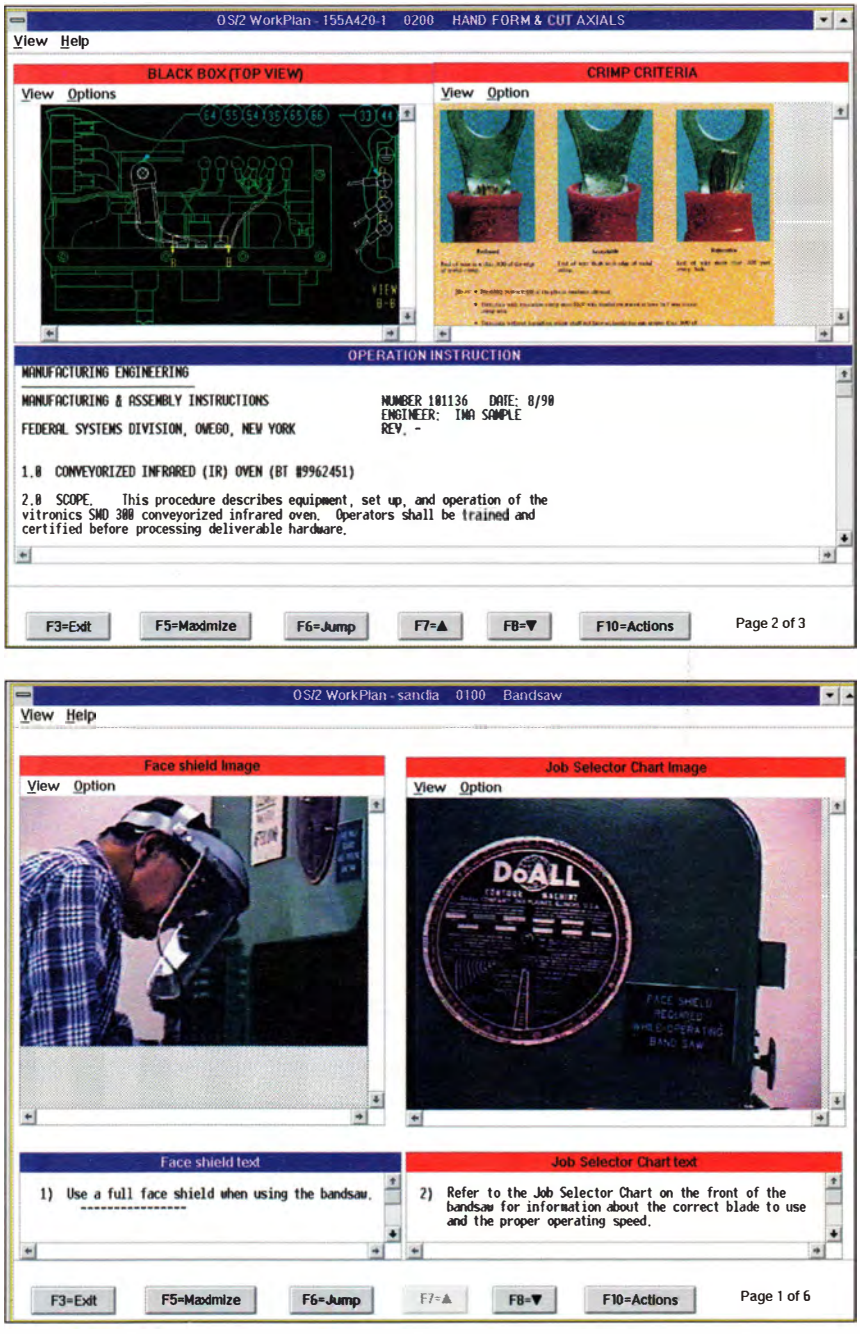
WorkPacket has all of the display capabilities of WorkBook plus significant additional function. WorkPacket allows the customer to associate order-specific information, such as Quantity, Priority, or Due Date, with the process plan. WorkPacket also allows data collected from the shop floor to be associated with the order (see Figure 6).

With WorkPacket, the process planner or production control person can merge manufacturing instructions with order specific requirements to create paperless shop orders. The data are captured with a bar code reader, magnetic strip reader, or keyboard to maintain serialized part control. The as built data, including the version of the process plan and operator signatures, are collected and can be reviewed as necessary.

WorkPacket provides product traceability. Electronic buyoffs or *signatures* provide product information control.

With WorkBook, WorkPlan, or WorkPacket you can print copies of a process plan or its various components (for example, specific operations, or the electronic buyoffs).

WorkPacket includes application program interfaces (API's) which enable customers to tie PMW into their manufacturing environments, including many legacy engineering and business planning systems.



Figures 4 and 5: Process Plans Can Include CAD Graphics, Scanned or Video-Captured Images Such As Photos and Drawings, and Text Such As Tables and Forms

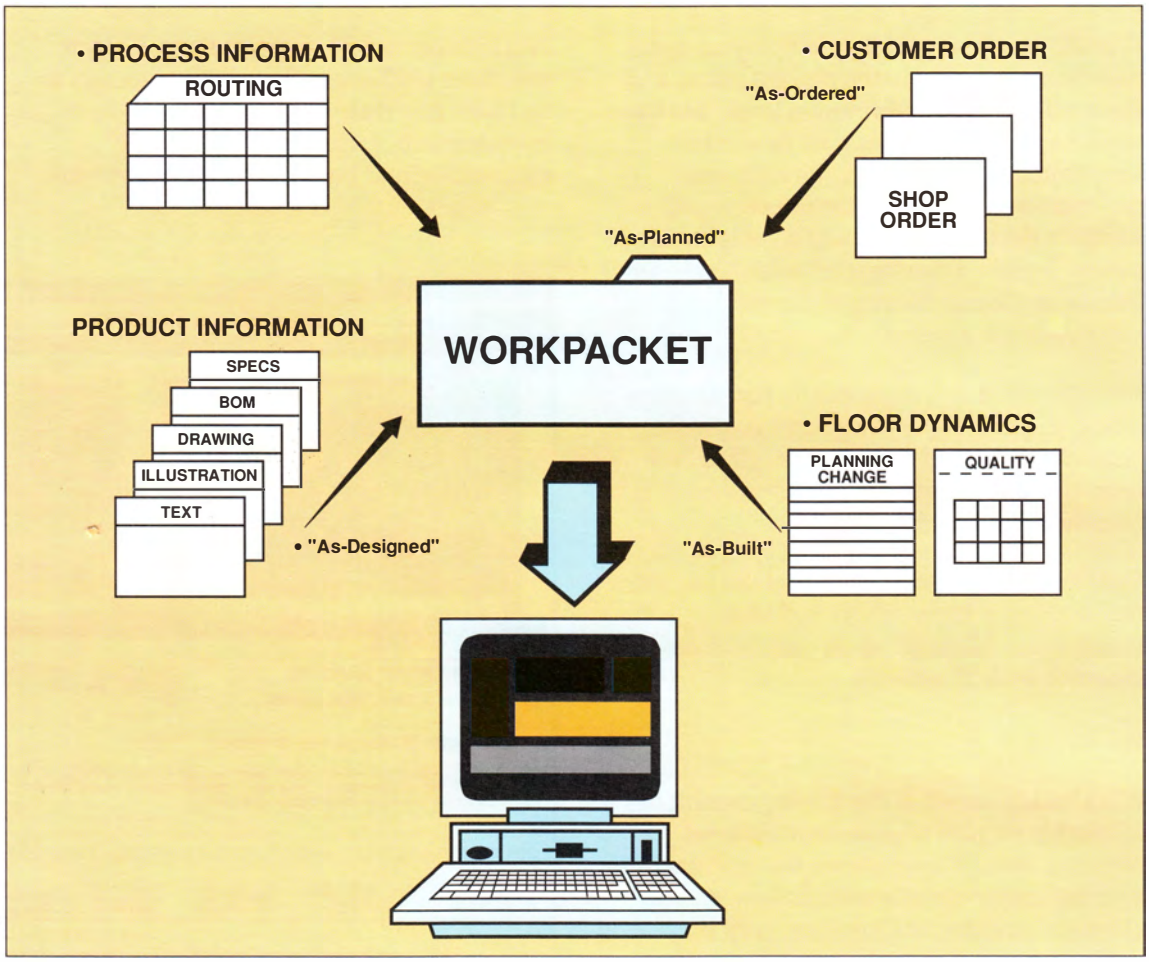


Figure 6. WorkPacket - The Multimedia Manufacturing Folder

PMW, running on OS/2, is used throughout the manufacturing process

Data collected with WorkPacket can be passed to other applications during the session. Also, the user can exploit OS/2 multitasking to drive other applications, such as statistical process control, labor claiming, or quality applications.

THE BENEFITS

Paperless Manufacturing WorkPlace provides many benefits for users in the manufacturing environment. With PMW, a user can:

- **Increase manufacturing planning effectiveness:** process plans, manufacturing instructions, reference documents, and work orders are created, organized, and maintained easily and accurately.
- **Increase shop floor operator effectiveness:** electronic workpackets provide well-organized and clearly illustrated manufacturing instructions.
- **Reduce plant floor errors:** current and accurate information is distributed to the plant floor electronically wherever and whenever it is needed.
- **Eliminate reports and forms:** manufacturing instructions, work orders, and other plant floor records are stored and distributed electronically.
- **Maintain accurate product records:** component identification, part substitution, and rework operations are collected in electronic work packets on a serialized part basis.
- **Improve product quality:** manufacturing instructions and instructional aids are improved and clarified easily, and redistributed quickly.

- **Provide accurate, up-to-date specifications:** the manufacturing operator on the plant floor receives the information necessary to ensure compliance with manufacturing practices, safety or government regulations, or process improvements.

THE REQUIREMENTS

PMW requires OS/2 Extended Edition 1.3 as the operating system and Distributed Automation Edition as the system enabler. Distributed Automation Edition provides a single, logical view of the shop floor network. In a distributed network it allows transparent communication, information sharing, and control among application and devices anywhere within the LAN.

Paperless Manufacturing WorkPlace consists of two releases. The first release is available now and includes WorkPlan, WorkBook, and WorkServer. The second release will include WorkPacket, as well as enhancements to WorkPlan, WorkBook and WorkServer.

CONCLUSION

Paperless Manufacturing WorkPlace provides the capability to improve planning, increase worker effectiveness, control shop orders, capture plant floor data, and provide as-built traceability and configuration control, quickly, accurately, and all in a paperless environment. It is an example of how OS/2 can be used to create a graphical/image intensive client-server application for manufacturing end users. These manufacturing solutions are enhanced greatly by OS/2 Presentation Manager, relational database, and multi-tasking capabilities. PMW exploits these capabilities to provide a state-of-the-art manufacturing application.

Kate Ferriter, *IBM Manufacturing Systems Products, 951 NW 51st Street, Boca Raton, FL 33431. Ms. Ferriter is an Advisory Planner in Production Operations Systems Development. She joined IBM in 1981 and has worked on application development assignments as well as manufacturing engineering and industrial engineering assignments in IBM Manufacturing. She holds a BS degree in Industrial and Systems Engineering from Georgia Tech.*

Performance

Performance Monitoring for OS/2: System Performance Monitor/2

Laura Camp

Vince Acosta

Performance tuning requires good data collection and monitoring tools

by Laura Camp and Vince Acosta

On any single computer system or network of systems, there are limited resources (such as CPU, RAM and disk) that need to be shared by all applications executing in that environment. The goal in managing and tuning performance is to maximize throughput and achieve acceptable response times within the constraints of the available resources. This means figuring out how to modify parameters, applications, and workloads such that the limited resources are used most effectively. To accomplish these tasks, performance data collection and monitoring tools are needed, as well as information on how to interpret the data and where to apply the interpretation in tuning the system.

This article will discuss System Performance Monitor/2, Version 1.0 (SPM/2). For information on how to tune the OS/2® system, read the IBM® Parameter and Tuning Guide for OS/2 EE and LAN Server, section 1.1.8, page 10. (This document is described further at the end of this article.)

OVERVIEW

SPM/2 consists of performance data collecting, monitoring, logging, reporting, and analyzing facilities executable in OS/2 Standard Edition or Extended Edition Version 1.2 or 1.3 environments. SPM/2 offers performance management functions for standalone systems as well as remote IBM OS/2 LAN systems. SPM/2 lets system administrators monitor system performance, analyze performance problems, perform load balancing, and manage network growth. In addition, SPM/2 helps verify performance objectives and fine-tune applications.

SPM/2 consists of the eight main components. Figure 2 illustrates the relationship among the first three of these.

- **Data Collection Facility (SPM/2 DCF)**

This is the fundamental component of the SPM/2 application. It interfaces with the OS/2 system and collects event data on the utilization of CPU, RAM and disk. Additionally, it provides information on file I/O and swap activity.

As illustrated in Figure 2, the Data Collection Facility communicates with other SPM/2 components or user applications via OS/2 Named Pipes. Only one application may access the Data Collection Facility at a time.

- **SPM/2 Monitor**

The SPM/2 Monitor is an application written to the SPM/2 API that provides a visual monitor of resource usage. It summarizes data from the Data Collection Facility and displays it in graph form in a Presentation Manager™ Window (see Figure 1).

- **Logging Facility**

The Logging Facility is another application that accesses the Data Collection Facility and logs the data to disk for post-processing. This logged data then is used as input to the Report Facility for generating formatted reports. Information provided in a report is at the dispatched process level. Summary reports are supported as are spreadsheet-compatible formats (tabular and delimited ASCII).

- **Memory Analyzer**

The SPM/2 Memory Analyzer (also known as Theseus) provides application developers with in-depth insight into OS/2 memory management. Its main feature is its capability to provide working set information per application, where working set is defined as that set of memory required for a given scenario.

- **Directory Analyzer**

This feature provides the capability to analyze disk capacity information (data file size and count).

- **Command Line Interface**

The full-function OS/2 Command Line interface to all SPM/2 Facilities.

- **Application Programming Interface (API)**

The API provides direct access to the data that is continuously collected by the Data Collection Facility.

- **SPM/2 Reference**

The SPM/2 Reference provides documented information on how to use SPM/2.

PERFORMANCE MANAGEMENT ENVIRONMENTS

SPM/2 can be used to monitor performance in the following four environments:

1. Standalone System: In this environment, the Data Collection Facility, the Monitor, and the Logging Facility all run on the same system. An example of a person who might use SPM/2 in the standalone environment is an application developer. The types of tasks performed might be design analysis, application and system profiling, and gaining an understanding of an application's use of the OS/2 system.

2. Remote OS/2 LAN Server(s): One or more OS/2 LAN Servers can be monitored remotely using SPM/2. The Data Collection Facility must be installed on each server to be monitored. A system from which one or more servers is being monitored is called a *managing system.* This system can be either an OS/2 LAN Requester or another OS/2 LAN Server, must have access to the named pipes on each server being monitored, and must have the Monitor or the Logging Facility executing.

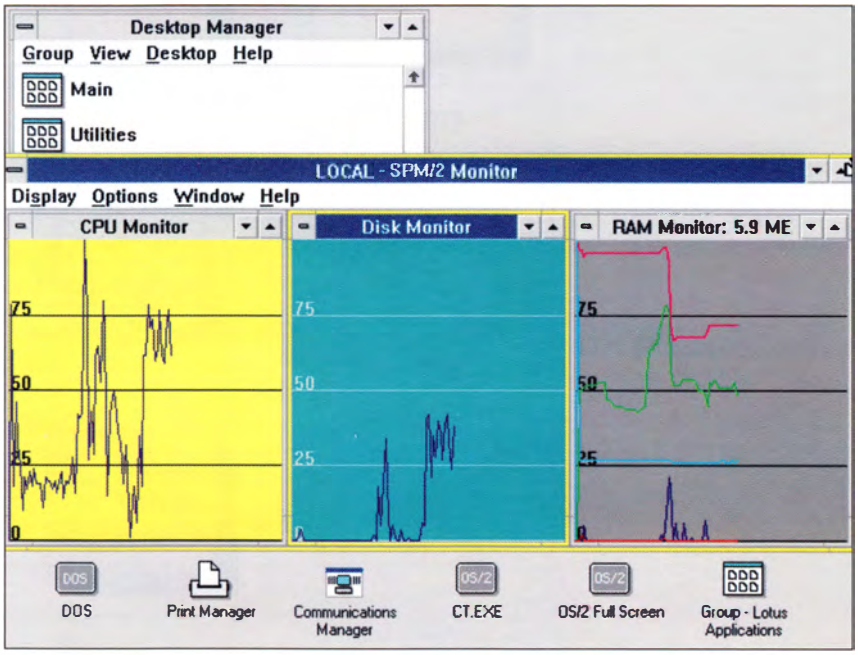


Figure 1. SPM/2 Monitor Showing Resource Usage

If more than one server is being monitored, a separate instance of the Monitor or Logging Facility must be started on the managing system for each server.

3. OS/2 LAN Workstations: OS/2 LAN Requesters can be monitored by executing the Data Collection Facility and the Log Facility on the requester and redirecting the output (via a network drive) to an OS/2 LAN Server for centralized processing.

4. Other Workstations: The remote monitoring of workstations other than OS/2 LAN Servers or Requesters (such as database servers) is possible through the use of the SPM/2 API. The SPM/2 API can be exploited by customer applications to manage these workstations remotely.

In any environment, the intrusiveness of a performance tool is an important concern. Since the data collected by SPM/2 is event-driven, the amount of data collected during any period of time will depend upon the level

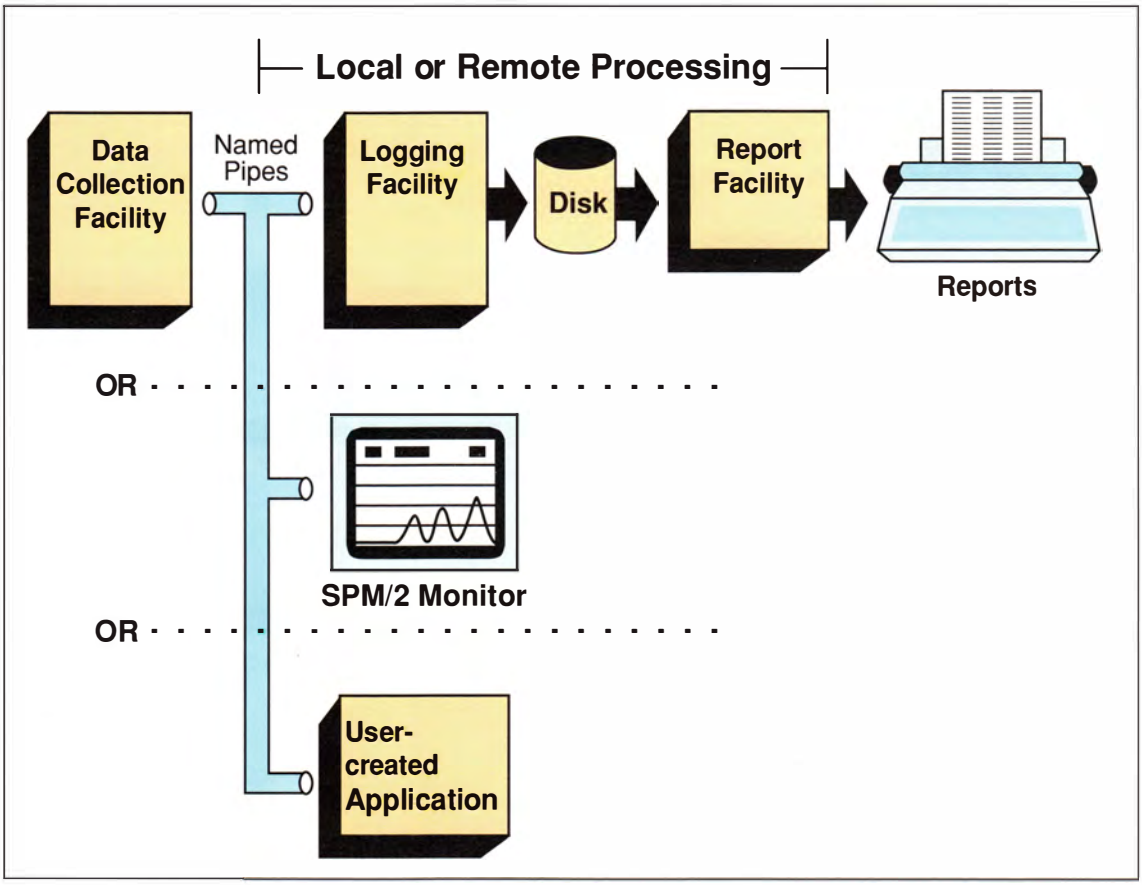


Figure 2. SPM/2 Overview

of system activity during that period. The SPM/2 Data Collection Facility typically utilizes less than three percent of the system CPU, minimizing the impact on a user's environment.

MORE DETAIL ON SPM/2

This section provides more detail on the function of SPM/2 by guiding you through a *demonstration*. We'll start by bringing up the Monitor program (see Figure 1). The Monitor can be started from the OS/2 command line by typing SPMMON, or by clicking on *SPM/2 Monitor* in the *SPM/2* group accessed from the Desktop Manager. In a standalone environment, the Data Collection Facility is started automatically by the Monitor or the Logging Facility. When SPM/2 begins execution, the data collected is accurate immediately. There is no calibration time.

The SPM/2 Monitor

As shown in Figure 1, the SPM/2 Monitor window contains three *sub-monitors* showing CPU, disk, and RAM utilization (including Swap In and Swap Out activity). Each monitor contains continuously graphed lines, indicating the percentage of a particular resource that is being used across time. Horizontal grids (labeled "0", "25", "50" and "75") are provided to help determine the percentage of a resource that is being used. The resource lines are updated at intervals specified in the "Time Periods" menu, accessible via "Options" on the Action Bar. Options in this menu are:

- **Viewing Period:** defines how much time is shown in each resource monitor.
- **Sampling Period:** defines how often the data are summarized and plotted as new points on the graph.

- **RAM Working Set Period:** defines the interval used in calculating theWorking Set. Working Set is the portion of RAM that has been *touched* during the specified working set period.

The CPU Monitor calculates CPU utilization through the use of a background process called IDLECPU. This process has the very lowest possible priority and runs only if no other tasks of higher priority are running in the system. Hence the amount of time used by processes of user interest will be 100 percent minus the percentage of time that IDLECPU runs. If other tasks in the system have the same priority as IDLECPU, these tasks will be time-sliced with IDLECPU.

The Disk Monitor measures the utilization of each physical disk in the system: read, write, and write-verify activity. It provides support for multiple SCSI, ESDI and ST-506 physical drive types, and for both FAT and HPFS file systems. Each physical disk is represented by a separate line on the Disk Monitor graph. Only IBM device drivers are supported since SPM/2 is dependent on OS/2 RAS systrace calls in the device drivers.

Information on DosRead and DosWrite activity against individual files can be obtained by measuring logical disk activity via the Logging Facility (see the section The SPM/2 Logging and Report Facilities later in this article).

The RAM Monitor

You'll notice in Figure 1 that the RAM Monitor title bar indicates the amount of memory available to OS/2. However, the number shown in Figure 1 may seem a bit peculiar — different than that provided by most memory boards! In this example, although 6 Mb is installed, only 5.9 Mb is indicated. This is due to the ROM BIOS being loaded into RAM when the system first powers up, thus decreasing the amount of memory available to OS/2. ROM BIOS uses 128Kb of RAM.

The RAM Monitor has five different lines. To understand these, let's take a look at the "Set Colors" screen, illustrated in Figure 3. (To access the "Set Colors" screen, select "Set Colors" from the "Options" menu in the Action Bar. For RAM Monitor settings, select "RAM Monitor" in the "Select Monitor" window of the "Set Colors" screen.)

Set Colors allows you to customize the colors of each submonitor: background color, grid color, and color of each line graphed. The first three lines listed for the RAM Monitor represent three types of memory utilization:

- **Used Memory**

Used memory is that portion of memory that has had something loaded into it or has been allocated to a program, since the system was powered up. This memory has been neither discarded nor released (freed).

Figure 3. SPM/2 Monitor - Set Colors Screen

- **Working Set Memory**

Working Set is a subset of Used Memory and is the portion of RAM that has been *touched* during a specified interval of time (the Working Set Period). Whether or not a portion of RAM has been *touched* is determined through the use of OS/2 system LRU (least-recently used) timestamps obtained by the SPM/2 Memory Analyzer. Working Set memory always is bounded by used memory and fixed memory.

- **Fixed Memory**

Fixed memory includes everything loaded in RAM that can neither be discarded nor

swapped to disk. This includes portions of the OS/2 operating system, device drivers, and any memory allocated as *fixed* by applications.

The last two RAM Monitor lines represent memory-swapping activity. The Swap-Out and Swap-In lines show the percentage of time the OS/2 system is busy swapping memory out to disk or in from disk, respectively. In both cases, this percentage includes time that the swap process is blocked by other processes or waiting for disk I/O.

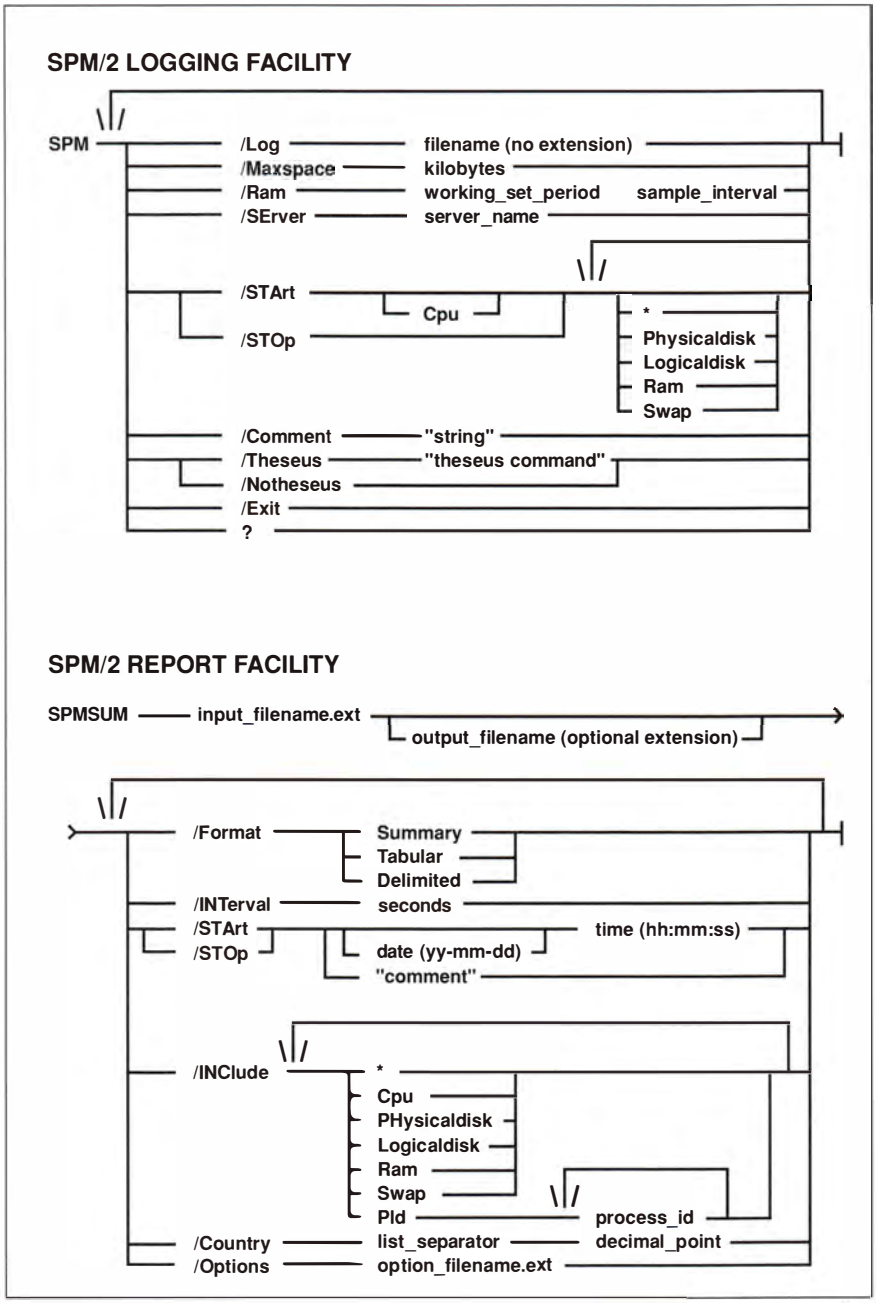


Figure 4. SPM/2 Logging and Report Facilities - Command Line Syntax

The SPM/2 Logging and Report Facilities

To obtain more detailed system performance information, the Logging Facility can be used. This facility enables you to see activity on a process basis, and to obtain read/write information on an individual file basis. Additionally, you can select specific resources for monitoring and can generate reports on subsets of the total data logged. (See Figure 4 for syntax of the Logging and Report Facilities.)

For example, if all that is needed is CPU and RAM activity for a local system, the following command will log this data to a file called OUTPUT:

```
SPM /L OUTPUT /START C L
```

The Logging Facility toggles between two files when writing to disk. In the example above, the file names would be: output.1 and output.2. When writing to one of the output files, SPM/2 has a system semaphore on that file. When approximately half the number of bytes specified in the /MAXSPACE parameter (defaults to 250 Kb) are written to that file, the file is closed, the semaphore is cleared, and a semaphore is obtained for the other file which is then opened and written to. This enables an application developer to obtain access to the *free* file for real-time processing.

As another example, if all resources have been logged, but you only want a summary report of the swap activity in 20-second intervals, the following command would be issued:

```
SPMSUM OUTPUT.1 /F SUMMARY /INC
SWAP /INT 20
```

The Logging Facility also can be used to execute Memory Analyzer (Theseus) commands remotely.

Figure 5 gives an example of a summary report. This report represents 1 minute and 28 seconds of activity on a system. During the test period (after initiating the Logging Facility), a disk-intensive program (CT.EXE) was started. This program alternates between intensive reading and writing to a file

```
**********************************************************************

CPU ACTIVITY

      -Process---     -----Execution-----     Number of     Secs. in
      Name    PID     Percent      Seconds    Dispatches    Interrupts

======================================================================

Interrupts  0000         8.9%       7.8551             0        0.0000
Other PIDs  0000         0.0%       0.0000             0        0.0000
 Dos3xBox   0002         0.0%       0.0000             0        0.0000
  Swapper   0003         0.6%       0.5516           113        0.0741
 ACSTRSYS   0004         0.0%       0.0032             5        0.0002
    ITDMN   0005         0.0%       0.0000             0        0.0000
  PMSHELL   0006         8.5%       7.5165           374        0.5865
  HARDERR   0007         0.0%       0.0000             0        0.0000
  PMSPOOL   0008         1.4%       1.2140           165        0.0768
   PMEXEC   0009         3.1%       2.7397           189        0.2119
      CMD   000E         1.4%       1.2188            58        0.1005
      CMD   0010         0.0%       0.0000             0        0.0000
     EZVU   0011         0.0%       0.0000             0        0.0000
RUMMINTRM   0012         0.0%       0.0000             0        0.0000
   ASPOOO   0013         4.1%       3.6659          1052        0.7412
 ACS3EINI   0014        10.2%       9.0708          1398        0.7918
      CMD   0017         0.0%       0.0000             0        0.0000
      CMD   0018         0.8%       0.6760            39        0.0535
      PE2   003D        11.3%      10.0114           414        0.7125
    WKSTA   003E         0.1%       0.0667            18        0.0198
 WKSTAHLP   003F         0.0%       0.0185             9        0.0016
 MUGLRQST   0041         0.0%       0.0000             0        0.0000
   SPMDCF   004E         2.7%       2.4244           208        0.1871
  THESEUS   0050         0.8%       0.7059            43        0.0512
  IDLECPU   0051        27.3%      24.1696          1044        2.4732
      SPM   0063         0.8%       0.7417           100        0.0793
     SEND   0065         2.8%       2.4685           322        0.3262
       CT   0066        15.2%      13.4832          1035        1.3677

    Total              100.0%      88.6015          6586        7.8551
**********************************************************************

PHYSICAL DISK ACTIVITY

Disk          Number of    Total --Elapsed Time per Request (seconds)-
 ID Operation  Requests  Sectors      Total  Average  Minimum  Maximum

======================================================================

x0080 Read          204     2111     6.1935   0.0304   0.0048   0.1225
      Write         747     1483    17.2702   0.0231   0.0055   0.0000
      Write/Verify    0        0     0.0000   0.0000   0.0000   0.0000
```

Figure 5. SPM/2 Summary Report (Continued)

```
*********************************************************************
LOGICAL DISK ACTIVITY

                    Number of      ----------------Bytes---------------
PID File name        Requests      Total    Average     Minimum   Maximum
=====================================================================
000E  D:\OS2\UTIL\CT.EXE-----------------------------------------------
      DosRead              3        475        158          64       347

0014  File handle x0003------------------------------------------------
      DosRead             12         93          8           5        14

003D  C:\IBMLAN\IBMLAN.INI---------------------------------------------
      DosRead              2       1521        761           1      1520

0063  SAMPREP.1--------------------------------------------------------
      DosWrite            19     125124       6585        4629      8102

0063  File handle x0004------------------------------------------------
      DosRead             38     125124       3293         534      4095

0065  File handle x0001------------------------------------------------
      DosWrite           114       4825         42           1       240

0066  E:\SPM2\CTTEST1.FIL----------------------------------------------
      DosRead            348     168000        483           0       512
      DosWrite           343     171584        500         176       512

*********************************************************************
RAM ACTIVITY

    Total System RAM:    7.875 Megabytes
  Working Set Period:    60 seconds
   Number of Samples:    9
                       -------Megabytes-------  --------Percent--------
                      Average Minumum Maximum  Average  Minimum  Maximum
                      ========================================================

   Allocated RAM (Used)  6.861   6.070   7.812    87.1%    77.1%    99.2%
 RAM In the Working Set  4.749   2.885   5.323    60.#%    36.6%    67.6%
              Fixed RAM  2.210   2.176   2.282    28.1%    27.6%    29.0%

*********************************************************************
SWAPPING ACTIVITY
                  Total     Total  --Elapsed Time per Segment (seconds)--
 Activity     Segements     Bytes      Total   Average   Minimum   Maximum
 ========================================================================

  Swap In            28    283679     2.5295    0.0903    0.0105    0.4072
  Swap Out            8    286720     0.9637    0.1205    0.0319    0.2415
```

Figure 5. SPM/2 Summary Report

(CTTEST1.FIL), and CPU-intensive bookkeeping. Hence CT.EXE generates CPU activity as well as physical and logical disk activity, as shown in the report.

The sections of the summary report are:

- **CPU Activity.** This section summarizes CPU activity for every process that was active in the system during the test period. Note that the CPU was 72.7 percent utilized — that is, 100 percent minus the 27.3 percent used by IDLECPU. CT.EXE utilized 15.2 percent of the CPU. Notice that the Data Collection Facility (SPMDCF) used only 2.7 percent of the CPU.

- **Physical Disk Activity.** There will be a section for each physical disk on the system. Since this activity is gathered at the device driver level, the hex disk ID is given instead of a drive letter. (These ID's start with x'0080'.)

- **Logical Disk Activity.** This information gives file read and write activity, independent of whether or not the physical disk was accessed. That is, there are times when data is read/written to a file, but the disk is not hit since the request could be satisfied by data in a cache. Note in the report that some files are named while others are simply represented by File Handle ID's. File handles are used for files that have been opened prior to starting the Data Collection Facility, since there is no way to obtain the actual file names. For files opened while the Data Collection Facility is active, names are obtained from the DosOpen function. In this example, the file SAMPREP.1 is one of the log files and CTTEST1.FIL is the file being accessed by CT.EXE.

- **RAM Utilization and Swapping Activity.** These are shown in the last two sections of the report.

As mentioned earlier, spreadsheet-compatible formats can be specified for reports. Figure 6 shows a graph created by Lotus® 1-2-3/G® by importing a delimited report from the SPM/2 Report Facility. Though the delimited report was generated from the same data that generated the report in Figure 5, it differed in that it was summarized in 1-second intervals. Each point in the graph represents one of these intervals. Notice that when CT.EXE disk activity is high, CPU activity is low, and vice versa. This is due to the program's switching between its read/write and bookkeeping functions.

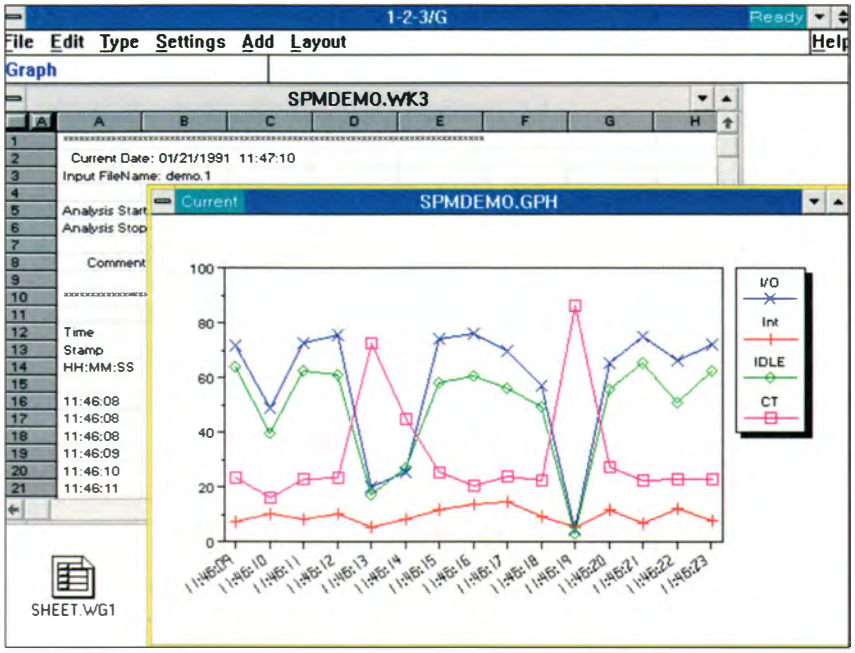


Figure 6. SPM/2 Delimited Report Imported into Lotus 1-2-3/G

The Memory Analyzer (Theseus)

The main feature of Theseus is its capability to provide Working Set information per application. Working Set analysis is facilitated by displaying the following system memory-related information:

- Memory usage analysis by application
- Loaded device driver information
- Free memory information
- Least recently used swappable/discardable segment information

The SPM/2 Memory Analyzer also can display the following system memory information:

- Virtual memory by specified selector
- Descriptor Table Entry information
- Local Descriptor Table information
- Loaded module information

- Process/thread information
- Per Task Data Area (PTDA) information
- Swapper information

The Directory Analyzer

The Directory Analyzer provides data file size and count information. This information can be reported at the following user-selectable levels:

- Directory
- Directory broken down by subdirectories
- Directory broken down by subdirectories and files
- Directory, subdirectory, or file, based on a hypothetical cluster length

The information provided in a Directory Analyzer report can include:

- Actual data size (in bytes) of a file
- Allocated data size (as a multiple of clusters) of a file
- Count of sized files
- Sum of actual data sizes of files included in a directory
- Sum of allocated data sizes of files included in a directory
- Sum of count of sized files included in a directory

DISTRIBUTED SUPPORT

As previously mentioned, SPM/2 can be used to monitor remote OS/2 LAN Servers. To do this, the Data Collection Facility must be started on the managed server. On the managing system, either the SPM/2 Monitor or the Logging Facility can be started from the OS/2 Command Line. The SERVER option is used to tell these facilities from where to get their data. Suppose the remote server name is MYSERVER. The command for the Monitor and the Logging Facility, respectively, would be:

```
SPMMON /S MYSERVER
SPM /L LOGFILE /SE MYSERVER
```

Servers that are monitored remotely do not need the SPM/2 Monitor. A Distributed Feature, which includes the Data Collection Facility, the Logging Facility, and all programs needed to run these components, can be purchased at a lower cost for such servers.

USER SCENARIOS

Following are four scenarios illustrating how SPM/2 can help users solve performance problems and answer performance questions.

Scenario #1

A large company reports a performance problem with one of their local production LAN systems. They have no idea why the system is running slow. Rather than waste time guessing at possible causes, SPM/2 is installed on the OS/2 LAN Server.

A log and summary report are generated, showing that the 12 Mb server is running out of RAM, resulting in swapping. There might be multiple ways of alleviating this problem. For example, a user program could be redesigned to use less RAM, or OS/2 system parameters could be modified to use less buffer space if possible. (For guidance in this area, see Parameter and Tuning Guide for OS/2 EE and LAN Server (1.2/1.3) described at the end of this article.) In this case, the customer makes the decision to buy more memory.

Scenario #2

A Communications Manager application is being developed for a large firm. The performance has just degraded, for no apparent reason. The only change has been the modification of a few lines of code, none of which seems to be performance related.

SPM/2 reveals that the application is disk-intensive, which was not expected for this application. Further investigation shows a communications allocation error that occurs repeatedly and is not being handled properly by the application. This allocation error is resulting in constant writing to ERROR.DAT by the OS/2 system. The information enables the programmer to correct the error handling routine.

Scenario #3

A small REXX program is written to test out the difference in response times between accessing an OS/2 database locally and accessing it remotely. Much to the user's surprise, the remote request finishes faster than the local request!

SPM/2 is loaded on the system and reports are generated for both the local and the remote cases. Analysis of the reports reveals the following:

- When the database is accessed locally, the CPU is the performance bottleneck. That is, CPU utilization is one hundred percent, while disk utilization is less than one hundred percent.

- When accessing the same database remotely, CPU utilization at the server becomes less than one hundred percent since just enough database application work is off-loaded to the requester. At this point disk utilization emerges as the server bottleneck. Database Manager is able to utilize the disk fully, resulting in better overall response time.

In this situation, SPM/2 provides the customer with the information necessary to rebalance the server's load, thereby obtaining more manageable utilization rates for CPU and Disk.

Scenario #4

A System Administrator for a local bank is informed that the tellers have been complaining about poor response from the bank's server. The System Administrator uses SPM/2 to monitor the system's performance remotely and address the problem. The Administrator starts data collection on each of the five tellers' systems, two program developer systems, and a network server, using preconfigured tool profiles. As the System Administrator monitors the seven systems remotely from the centralized console, reports are generated for each system.

Within minutes it is obvious from the reports that one of the program developers, a relatively new employee, is over-utilizing one of the server's physical drives, which the tellers happen to share. The reports reveals a seventy percent utilization on one of the developers' drives, which also happens to be server supported.

The customer redirects the developers' disk-intensive application debugging efforts to a different off-line server to avoid future performance impacts to the business. The tellers' performance is restored and business returns to normal.

CONCLUSION

System Performance Monitor/2 provides powerful capabilities for collecting, monitoring, logging, reporting , and analyzing system performance data. These capabilities enable both system administrators and application programmers to manage OS/2 system performance better, and thereby protect their OS/2 and PS/2® investments, as well.

PARAMETER AND TUNING GUIDE FOR OS/2 EE AND LAN SERVER

The publication Parameter and Tuning Guide for OS/2 EE and LAN Server (1.2/1.3) provides performance tuning guidance and information for Versions 1.2 and 1.3 of the IBM Operating System/2® Extended Edition (EE) and the IBM OS/2 Local Area Network (LAN) Server and Requester programs. Its purpose is to enable users of the OS/2 system to tune their systems and applications for optimum performance by providing background information on OS/2 system structure and performance-related parameters, as well as tuning tips and guidelines.

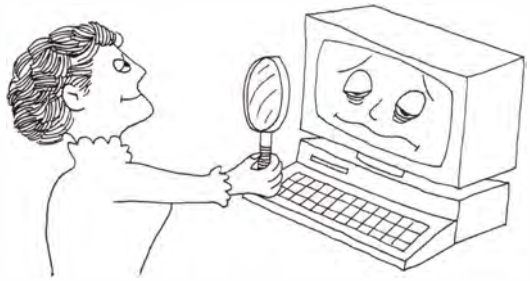
OS/2 topics and system components covered are:

- "How to" do performance tuning
- File Systems
- Communications Manager
- LAN Server and Requester
- Database Manager
- Application Design and Development Guidelines
- DOS Compatibility Mode Hints

To obtain this document, call your local branch office and submit a request via the IBM System Library Subscription Service (SLSS). Specify order number G33F-9437.

Laura Camp, *Advisory Programmer, IBM Austin Programming Center. Ms. Camp is currently responsible for communicating OS/2 performance information to the field. She joined IBM in 1976 where she worked on the design and development of DisplayWriter products. In 1986 she joined the OS/2 Database Manager development team, followed by two years in OS/2 Extended Edition product support, where she was involved in training customers on the use of the Database Manager. Ms. Camp holds a BA from Hope College, and majored in Mathematics with Computing Emphasis.*

Vince Acosta, *Advisory Planner, IBM Austin Programming Center. Mr. Acosta has been with IBM Austin since graduating from the University of Texas at El Paso in June 1976 with a BS in Electrical Engineering. Transferring from the hardware side of the house, he has spent the last five years in the Programming Center performing programming, team leading and technical liaison tasks related to the OS/2 EE Communications Manager. Mr. Acosta's current role is that of a Technical Planner for the SPM/2 product in the OS/2 Distributed Systems Management area.*



Performance

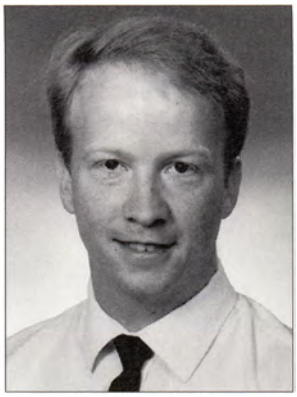
An OS/2 High Resolution Software Timer

by Derek Williams

OS/2® allows users to measure time intervals with a 32 millisecond resolution. Traditionally, developers requiring a higher-resolution measurement have purchased special-purpose hardware. This article describes a device driver which can measure time intervals with about a one-microsecond resolution and which does not require additional hardware. Complete source code for the device driver is provided, which can serve as a good introduction to OS/2 1.x device drivers and to the timing hardware contained on all PCs and PS/2® systems.

In software products, as in all of life, time often is of the essence. Measuring and tuning program performance is an important and often critical process in developing quality software. OS/2 provides APIs for measuring time intervals (DosTimerAsync and DosTimerStart) and assigning time limits to various system operations (for example, limits to blocking intervals on DosSemWait calls). However, OS/2 limits the resolution, or granularity, of these intervals to the 32 millisecond range and provides no support for measuring intervals with a finer resolution. While a 32 millisecond granularity is fine for most performance measurements, plenty of exceptions have been identified. For example, in the bank check processing environment, customer-written programs must be tested to ensure they execute within 18 milliseconds. This is due to the timing constraints of the hardware on which these programs run.

The means for measuring software time intervals is simple: collect timestamps for various start and stop points, and then factor out the overhead of gathering these time-stamps in order to generate a net, elapsed time. The difficult part is finding a tool which can provide adequate timestamps. A number of hardware timers are available which can be purchased and installed in AT® and PS/2 systems, and then used with some interface software to generate timestamps. The Timer Device Driver (TDD) discussed in this article provides an alternative to purchasing additional hardware: it uses the timer chips already available in AT class machines to provide high-resolution timestamps.



Derek Williams

This device driver uses standard PS/2 timers for high-resolution measurements

TIMING HARDWARE IN THE OS/2 ENVIRONMENT

All machines which run OS/2 include two timer chips:

1. An Intel 8253 (or 8254): This programmable timer is driven by a 1.19 MHz oscillator and provides three separate 16-bit counters. All counters operate on 840 nanosecond *ticks* (840 nanoseconds = 1 / 0.00119 cycles per nanosecond). The output from one of these counters (counter 0) is connected to the IRQ 0 interrupt request line on the interrupt controller. Each time counter 0 *times out*, interrupt 8 occurs. The BIOS default for this timeout is every 55 milliseconds; 55 milliseconds = 65,535 (counter timeout value) / 1190 cycles per millisecond. Since all IBM® PC compatible machines include this chip, the PC BIOS relies upon this interrupt to maintain time of day and to service various hardware devices. All PC-DOS programs which require timer support use some derivative of BIOS interrupt 8.

One of the nice features of the 8253 is that it has counter registers which can be read via I/O ports. Not only does counter 0 allow the

interrupt 8 service routine to keep a running total of 55 millisecond intervals, but any program can read a register for counter 0 to know how many nanoseconds have elapsed since the last interrupt, giving a complete high-resolution time of the form:

Total Nanosecs = 840 * ((Number of int 8s * 65,535) + (Ticks since last int 8))

2. A Motorola MC146818: This chip contains the real time clock/calendar circuits and CMOS RAM found on AT class machines and PS/2s. Since this chip usually is not found on 8086 and 8088 machines (unless purchased separately), PC-DOS does not depend on this chip for hardware servicing. This chip is wired to produce IRQ 8, which causes interrupt 70h to be called at regular intervals. This regular interval is initialized to 32 milliseconds.

The MC146818 itself maintains running time totals (seconds, minutes, hours, days, months, years), but does not provide support for reading tick values within the interrupt period.

The counters on the 8253 timer, together with the running time total maintained by the interrupt 8 service routine provide support not only for measuring long time intervals, but for measuring intervals with an 840 nanosecond resolution. This works well for programs running under DOS, since DOS relies upon the 8253 timer support provided by the BIOS. Unfortunately, OS/2 uses the Motorola chip for all of its timing and actually prevents use of interrupt 8 for keeping an accurate running time total. Since the 8253 interrupt 8 is not used by OS/2 it is masked off in protect mode at the interrupt controller to avoid the overhead in servicing two timers on two different cycles.

GAINING THE BEST OF BOTH TIMERS

When looking at these two timer chips and how they are used in an OS/2 environment, we see that each offers its own problems and solutions. OS/2 relies on the Motorola chip to generate interrupts every 32 milliseconds. However, there is no way to read the Motorola chip to determine the time since the last interrupt so as to get a resolution better than 32 milliseconds. Programs can read the 8253 to get a *time since the last interrupt* value, but by masking off the 8253-generated interrupt in protect mode, OS/2 prevents any service routine from keeping a valid running total. The TDD takes the best of both timers, enabling the user to keep long time totals which have very fine resolution.

To do this, the Timer Device Driver:

1. Hooks into the interrupts generated by the MC146818 to keep accurate running totals.

2. Sets up the 8253 to get reliable information from its tick counting registers.

3. Provides an interface to retrieve time stamps.

OS/2 allows device drivers to hook into the MC146818 interrupt by way of the SetTimer Device Helper. Thus, a service routine can retrieve timer tick counts at each interrupt and keep running totals.

For each interrupt generated by the MC146818, the TDD reads the current tick count from the registers of the 8253 to keep a running total. This requires that the period of the 8253 be longer than that of the Motorola chip (i.e., the time interval before the counter rolls over must be longer than than 32 milliseconds). All 8253 counters, however, are initialized to modes such that the tick counts roll over before 32 milliseconds. The TDD reprograms one of the 8253 counters to prevent it from rolling over between interrupts.

Care is required when reprogramming a counter to ensure that other users of the counter are not affected and that the counter used will not be reprogrammed by other applications. After looking into the normal modes and uses of the 8253's counters on OS/2 systems, an obvious choice is to reprogram counter 0 from mode 3 to mode 2 using an initial count of zero. The PC and PS/2 Technical Reference Manuals explain the counters and modes; in short, switching counter 0 this way does not affect any programs running in DOS (real) mode, and provides some security in that almost no DOS or OS/2 programs reprogram counter 0.

Applications must be able to read the time stamps from the driver; device driver read requests are suitable for this purpose. Each time a request is made to return a timestamp, the time since the last interrupt is added to the current time total to return an *up-to-the-microsecond* timestamp.

CODE LISTINGS

Before looking at the device driver itself, it helps to look at some sample code showing how to use the device driver.

Sample Code to Use the Timer Device Driver

Figure 1 shows some sample code to use the timer device driver. Before using the device driver, an application must open it and get a file handle to read from it. The timer device driver is set up to respond to OPEN and CLOSE requests, and to return a current timestamp on each READ request. We use OS/2 APIs to open, close, and read from the device driver. We could use C runtime functions such as open(), read(), and close(), but the OS/2 calls have slightly less overhead.

Each read from the device driver returns a fixed number of bytes in the format described by the TIMESTAMP structure. The timestamp includes the millisecond and nanosecond portions of the time stamp as well as a device driver version number. (When shipping a device driver as part of a product, it's a good idea to have the device driver provide some indication of level to its users, although we can do without level checking in our test program.) You can gather start and stop timestamps around any code you wish to measure, then subtract the stop time from the start time to generate an elapsed time. However, when measuring elapsed times, be sure to:

1. Factor out overhead.

As Heisenberg stated long ago, whenever you measure something, you must take into account the effect of your measuring tools. In this case, an application using the TDD must factor out the time taken to gather timestamps. The easiest way to do this is to read a start timestamp then immediately, with no intervening code, read a stop timestamp. This elapsed time is the overhead incurred when

```
typedef struct {
   ULONG millisecs;
   ULONG nanosecs;
   ULONG version;
} TIMESTAMP;

TIMESTAMP StartTime, StopTime;
ULONG msecs, nsecs;
                                  :
                                  :

     if (rc=DosOpen( "TIMER$", &TimerHandle, &ActionTaken, (ULONG) 0L,
                     FILE_NORMAL, FILE_OPEN, OPEN_SHARE_DENYNONE, (ULONG) 0L ))
        exit(printf("\nDevice Driver not opened, rc= %d\n", rc));

     if ((rc=DosRead( TimerHandle,        /* Read the start timestamp    */
                      (PVOID) &StartTime,
                      sizeof(StartTime),
                      &BytesRead         )) || (BytesRead < sizeof(StartTime)))
        exit(printf("\nError on DosRead, rc=%d,  Bytes=%d\n", rc, BytesRead));
                                  :
```

Figure 1. Sample Code to Use the Timer Device Driver (Continued)

```
                                :          /* Code to measure here...      */
                                :

   /* Call DosRead again to get the stop timestamp in StopTime...          */

   /* Calculate millisecs and nanosecs - borrow a millisec if needed:      */

   msecs = StopTime.millisecs - StartTime.millisecs;
   if (StartTime.nanosecs > StopTime.nanosecs) {
      nsecs = (1000000 + StopTime.nanosecs) - StartTime.nanosecs;
      msecs--;
   } else
      nsecs = StopTime.nanosecs - StartTime.nanosecs;
```

Figure 1. Sample Code to Use the Timer Device Driver

reading timestamps. This overhead typically is around half a millisecond; most of this overhead is the time taken by DosRead to gather data from the device driver.

2. Eliminate background activity.

OS/2's multitasking capabilities can be wonderful for getting your work done. However, when measuring elapsed times, you should avoid running other jobs which can steal time away from the program you are measuring. Be careful that no other programs are running while gathering your timing measurements and do as much as you can to avoid *hidden* activity such as swapping and background communications.

3. Take averages.

While you can do much to reduce the effects of *foreign* code stealing time from your program, you never can lock out interrupts or OS/2's own scheduler completely. For this reason, you should, whenever possible, take an average of several timing measurements. You may need to throw out *anomalies* caused by other tasks stealing time away from your program. At any rate, by having several similar measurements, you can be more certain that each measurement is accurate and not affected by other tasks.

Timer Device Driver Source Code

Figure 2 is a complete listing of the TDD source code. The code is written in assembler for MASM. It begins with various constant and data declarations, including:

1. A definition of the packet returned by the device driver on a Read request.
2. Definitions for OS/2 device driver request packets.
3. Constants for the OS/2 DevHlp codes and for the I/O ports of the 8253 chip.
4. A private data area used by the device driver.

Following this, you can see, in order, the standard device driver components:

- Device driver header
- Strategy routine
- Interrupt routine
- Initialize routine

The strategy routine is used to handle all request packets passed by OS/2 to the device driver. In general a strategy routine can support a wide range of requests; the TDD only needs to support the following requests:

- Initialize driver
- Device open
- Device close
- Read from device

For the initialize request, we use our Initialize routine; this is the last routine in the listing. This routine runs when the device driver is first loaded, then is removed from memory when it completes. The initialize routine does the following:

1. Saves the address to the DevHlp interface for future use.

2. Sets the 8253 counter 0 to run in mode 2 with an initial count of 0.

3. Hooks into the MC146818 interrupt using the SetTimer device helper.

In this case, our *interrupt* routine is a handler for the SetTimer DevHlp. The INTERRUPT routine is called at each interrupt of the MC146818.

When the first user opens the device driver, the OPEN routine resets the running timestamp to zero. Then it gets the current tick count from the 8253 timer 0 and makes this the Last8253 tick count. Each time an interrupt or READ request comes to the device driver, the Last8253 tick count is used to build an *up-to-the-microsecond* timestamp. The READ routine gets the current tick count from the 8253 timer 0, uses the Last8253 tick count to

```
;*************************************************************************
;          TIMER$ - Timer Device Driver for High-res timings
;*************************************************************************

ReadData              struc         ; Data passed on each Read request
    Read_Millisecs    dd   ?        ; The 'time stamp' - millisecs and
    Read_Nanosecs     dd   ?        ;  nanosecs
    Read_Level        dw   ?        ; Device Driver version information
    Read_Revision     dw   ?
ReadData              ends

RequestPacket         struc         ; Common header for all requests
    RP_Length         db   ?        ; Length of the request packet
                      db   ?        ; Block devices only
    RP_CommandCode    db   ?        ; Command Code - See table
    RP_ErrorCode      db   ?        ; OS/2-defined error codes
    RP_Status         db   ?        ; Status flags:
                                    ;  RP_StatusError, RP_StatusDone
                      dd   ?        ; Reserved
                      dd   ?        ; Queue Linkage - not used here
RequestPacket         ends

RP_StatusError        equ  80h      ; RP_Status error bit
RP_StatusDone         equ  01h      ; RP_Status done bit

RequestPktInit        struc         ; Cmd=0 - Initialize driver
                      db 13 dup(?)  ; Request header
    RPO_NumberUnits   db ?          ; Used to cancel init
                                    ; Pointer to DevHlp-on init return has:
    RPO_EndOfCode     dw ?          ;  End of Code pointer
    RPO_EndOfData     dw ?          ;  End of Data pointer
    RPO_InitArgs_Ptr  dd ?          ; Ptr to Init args
                      db ?          ; Block devices only
RequestPktInit        ends

RequestPktRead        struc         ; Cmd=4 - Read from device
```

Figure 2. Source Code for the Timer Device Driver - TIMER.ASM (Continued)

```
                        db 13 dup(?) ; Request header
                        db ?         ; Media - block devices only
    RP4_TransferAddr    dd ?         ; Phys address of read buffer
                                     ;  for PhysToVirt
    RP4_BytesRequested  dw ?         ; Number of bytes to read
RequestPktRead          ends

DevHlp_PhysToVirt       equ     15h  ; Device Helpers used by the driver
DevHlp_SetTimer         equ     1Dh
DevHlp_UnPhysToVirt     equ     32h

i8253CounterZero        equ     40h  ; 8253 chip ports and commands
i8253CtrlZeroOrTwo      equ     43h
i8253CmdReadCtrZero     equ     0
i8253CmdInitCtrZero     equ     34h

Read8253        MACRO               ; Gets current 8253 timer 0 value in CX
        mov     al,i8253CmdReadCtrZero      ; Request counter latch
        out     i8253CtrlZeroOrTwo,al
        in      al,i8253CounterZero         ; Read the LSB,
        mov     cl,al                       ;  save it, then
        in      al,i8253CounterZero         ; Read the MSB,
        mov     ch,al                       ;  and save it
                ENDM

data                    SEGMENT   public    'DATA'
;*************************************************************************
;                   TIMER Device Driver header
;*************************************************************************

TimerHeader             label   byte
    Next_DD_Ptr         dd   -1         ; Pointer to next driver
    Device_Attribute    dw   1000100010000000B ; Driver attributes:
                                        ;  Character, Open/Close, OS/2 driver
    Strategy_Offset     dw   Strategy   ; Offset of Strategy routine
                        dw   -1         ; IDC not used here
    Dev_Drvr_Name       db   'TIMER$  '; Driver name
                        db   8 dup(0)   ; Reserved - must be zero

;*************************************************************************
;              Data areas used by Strategy and Interrupt
;*************************************************************************

DevHlp_Ptr              dd   ?          ; Set at Init request
DD_ReadData             ReadData <0,0,1,0> ; Data to return to caller
ReadDataLen             equ  $ - DD_ReadData
UserCount               dw   0          ; Number of active users
Last8253                dw   ?          ; 8253 tick count at last interrupt
```

Figure 2. Source Code for the Timer Device Driver - TIMER.ASM (Continued)

```
;*************************************************************************
;                        Command (Request) List
; The RP_CommandCode value is used to dispatch to the correct routine
;*************************************************************************

CmdList         label     word
                dw        Initialize      ;  0 = Initialize driver
                dw        Error           ;  1 = Media Check
                dw        Error           ;  2 = Build BPB
                dw        Error           ;  3 = Not used
                dw        Read            ;  4 = Read from device
                dw        DummyRet        ;  5 = Non-destructive read
                dw        DummyRet        ;  6 = Return Input Status
                dw        DummyRet        ;  7 = Flush Input Buffers
                dw        DummyRet        ;  8 = Write to device
                dw        DummyRet        ;  9 = Write with verify
                dw        DummyRet        ; 10 = Return Output status
                dw        DummyRet        ; 11 = Flush Output buffers
                dw        Error           ; 12 = Not used
                dw        Open            ; 13 = Device open
                dw        Close           ; 14 = Device close
MaxCmd          equ       ( $-CmdList ) / TYPE CmdList
LastData        equ       $
data                      ENDS

code                      SEGMENT   public    'CODE'
                          assume cs:code, ds:data

;*************************************************************************
;                      STRATEGY Entry Point
;*************************************************************************

Strategy        PROC      far
          cmp   es:|bx|.RP_CommandCode,MaxCmd ; Command in our jump table?
          jbe   JumpCmd                     ;  Yes: Use the table
          call  Error                       ;  No:  Use the Error routine
          jmp   short exit                  ;       then exit

JumpCmd:  mov   al, es:|bx|.RP_CommandCode  ; Get the command code,
          cbw                               ;  convert to a word,
          mov   si,ax                       ;  make it an index...
          shl   si,1                        ;  into our word-wide table,
          call  CmdList |si|                ;  and away we go...
exit:     ret                               ; Far return to OS/2
Strategy        ENDP

;*************************************************************************
;                    Read:  Handle a Read request
; Register usage:
;     CX - TimerValue - current 8253 tick count
;*************************************************************************
```

Figure 2. Source Code for the Timer Device Driver - TIMER.ASM (Continued)

```
Read            PROC
        cli                                 ; Disable interrupts
        Read8253                            ; Read the current 8253 timer
                                            ;  0 value into CX
         call UpdateTimeStamp               ; Update the running timestamp

    ; Copy the Timestamp to the Read transfer address:

         cmp  es:|bx|.RP4_BytesRequested,ReadDataLen
         jl   SetError                      ; Caller's buffer is too small
         push es                            ; Save packet ptr
         push bx
         mov  ax,word ptr es:|bx|.RP4_TransferAddr+2
         mov  cx,es:|bx|.RP4_BytesRequested ; Length of segment
         mov  bx,word ptr es:|bx|.RP4_TransferAddr
         mov  dh,1                          ; 1 = Store result in ES:DI
         mov  dl,DevHlp_PhysToVirt
         call DevHlp_Ptr                    ; Call the PhysToVirt DevHlp
         jc   DevHlpError                   ; Carry set = DevHlp failed

         mov  si,offset DD_ReadData         ; DS:SI -> Timestamp
         mov  cx,(ReadDataLen / 2)          ; Number of words to move
         cld                                ; Move forward
         rep  movsw                         ; Move the data
         sti                                ; Reenable interrupts
         mov  dl,DevHlp_UnPhysToVirt        ; Req'd Only if LOADALL used
         call DevHlp_Ptr                    ; Cal the UnPhysToVirt DevHlp
         pop  bx                            ; Restore packet ptr
         pop  es
         mov  es:|bx|.RP4_BytesRequested,ReadDataLen
         jmp  short GetOut                  ; Skip over error case

DevHlpError:
         pop  bx                            ; Restore packet ptr
         pop  es

SetError: mov  es:|bx|.RP4_BytesRequested,0  ; Nothing read

GetOut:  sti                                ; Reenable interrupts
         or   byte ptr es:|bx|.RP_Status,RP_StatusDone
         ret
Read            ENDP
;******************************************************************************
;                   Open:  Handle an Open request
;******************************************************************************

Open            PROC
         cli                                ; Disable interrupts

; If this is the first user opening, initialize timestamp and Last8253:
```

Figure 2. Source Code for the Timer Device Driver - TIMER.ASM (Continued)

```
        cmp   UserCount,0                   ; First user opening?
        jnz   AddnlUser                     ;   No: Init not needed
        mov   word ptr DD_ReadData,0        ; Faster than ES save/restore
        mov   word ptr DD_ReadData+2,0      ;  for STOSW
        mov   word ptr DD_ReadData+4,0
        mov   word ptr DD_ReadData+6,0
        Read8253                            ; Get 8253 timer 0 into CX
        mov   Last8253,cx                   ; Save for next interrupt

AddnlUser:
        inc   UserCount                     ; One more user
        sti                                 ; Done with our critical sect
        or    byte ptr es:[bx].RP_Status,RP_StatusDone
        ret
Open    ENDP

;*************************************************************************
;                Close:  Handle a Close request
;*************************************************************************

Close   PROC
        cli                                 ; Enter 'critical section'
        cmp   UserCount, 0                  ; Any users?
        jz    NoUsers                       ;  No: Count is already zero
        dec   UserCount                     ; Yes: One less user
NoUsers:sti                                 ; Done with our critical sect
        or    byte ptr es:[bx].RP_Status,RP_StatusDone
        ret
Close   ENDP

;*************************************************************************
;              Error:  Handle an unsupported request
;*************************************************************************

Error   PROC
        mov   byte ptr es:[bx].RP_ErrorCode,3; Unknown command
        or    byte ptr es:[bx].RP_Status,RP_StatusError+RP_StatusDone
        ret
Error   ENDP

;*************************************************************************
;         DummyRet:  Handle a required but unused request
;*************************************************************************

DummyRet        PROC
        or    byte ptr es:[bx].RP_Status,RP_StatusDone
        ret
DummyRet        ENDP
```

Figure 2. Source Code for the Timer Device Driver - TIMER.ASM (Continued)

```
;*************************************************************************
;          Interrupt - Device Driver Interrupt-time routine
;           Called on each OS/2 clock tick (MC146818 chip),
;                    via the SetTimer DevHlp.
; Register usage:
;     CX - TimerValue - current 8253 tick count
;*************************************************************************

Interrupt          PROC      far

    ; If there are no users, simply return:
         pushf                                ; Save flags just in case
         cmp       UserCount,0                ; Anyone using us?
         jz        NoUser                     ;   No: Get out
         push      ax                         ; Save all registers we use
         push      bx
         push      cx
         push      dx

         cli                                  ; Clear interrupts for read
         Read8253                             ; Get 8253 timer 0 into CX
         sti                                  ; Reenable interrupts
         call      UpdateTimeStamp            ; Update the running timestamp
         pop       dx                         ; Restore saved registers
         pop       cx
         pop       bx
         pop       ax

NoUser:  popf                                 ; Restore flags just in case
         ret                                  ; Far return to OS/2
Interrupt          ENDP

;*************************************************************************
;           UpdateTimeStamp - Update the running timestamp
;                  Called by Read and Interrupt
; On call:
;     CX - TimerValue - current 8253 tick count
; On return
;     DD_ReadData is updated to reflect the current timestamp.
;     Last8253 is updated for the current 8253 tick count.
;     AX and DX are modified; BX is preserved
;*************************************************************************

UpdateTimeStamp    PROC
         push      bx                                 ; Save BX

    ; Determine the number of nanoseconds passed since the last timestamp
    ;  update (NanosecsDelta), as follows:
    ;
    ;       IF Last8253 >= TimerValue THEN
    ;          NanosecsDelta = (Last8253 - TimerValue) * 840;
```

Figure 2. Source Code for the Timer Device Driver - TIMER.ASM (Continued)

```
;       ELSE              /* Timer wrapped: */
;          NanosecsDelta = ('FFFF'X - TimerValue + Last8253) * 840;

         mov       ax,Last8253              ; Get Last8253 in AX
         cmp       ax,cx                    ; Last8253 >= TimerValue ?
         jae       NoWrap                   ;  YES: don't adjust for wrap
         sub       ax,cx                    ; Last8253 - TimerValue
         dec       ax                       ;  Adjust to handle wrap
         jmp       short CvtNsecs           ; Go convert to nanosecs
NoWrap:  sub       ax,cx                    ; Last8253 - TimerValue
CvtNsecs:mov       dx,840                   ; Cvt increment to nanosecs
         mul       dx                       ;  by multiplying by 840

; Save TimerValue as Last8253 for the next interrupt: Last8253=TimerValue

         mov       Last8253,cx              ; Done with old Last8253

; Update the running timestamp and normalize the millisecond and
;  nanosecond portions, as follows:
;
;    DD_ReadData.Read_Nanosecs = DD_ReadData.Read_Nanosecs +
     NanosecsDelta;
;    IF DD_ReadData.Read_Nanosecs >= 1000000 THEN   /* 1000000 = F4240h */
;    DO;
;    /*     Normalize the millisecond / nanosecond counters:        */
;       DD_ReadData.Read_Millisecs = DD_ReadData.Read_Millisecs +
;                                    (DD_ReadData.Read_Nanosecs / 1000000);
;       DD_ReadData.Read_Nanosecs = DD_ReadData.Read_Nanosecs % 1000000;
;    END;

     add       word ptr DD_ReadData.Read_Nanosecs,ax    ; Add in the
     adc       word ptr DD_ReadData.Read_Nanosecs+2,dx  ; increment
     cmp       word ptr DD_ReadData.Read_Nanosecs+2,0Fh ; Check high word
     jb        UpdateExit                               ;  lower: exit
     ja        Normalize                                ;  higher: normlze
     cmp       word ptr DD_ReadData.Read_Nanosecs,4240h ; Check low word
     jb        UpdateExit                               ;  lower: exit

Normalize:
  ; We need the result and remainder from dividing Read_Nanosecs by
  ; 1000000.  The 80286 does not support double-word divisors, so we
  ; must stage the division.  1000000 is 64 * 15625, so we can get our
  ; result and remainder as follows:
  ;    Divide Read_Nanosecs by 15625
  ;       Save the remainder
  ;    Divide the previous result by 64 to get the final quotient
  ;       Remember the remainder (we can use masks and shifts to
  ;       determine remainder and result).
  ;    Multiply the current remainder by 15625 and add in the saved
  ;       remainder to get the final remainder.
```

Figure 2. Source Code for the Timer Device Driver - TIMER.ASM (Continued)

```
     mov    dx,word ptr DD_ReadData.Read_Nanosecs+2  ; Get high portion
     mov    ax,word ptr DD_ReadData.Read_Nanosecs    ; Get low portion
     mov    bx,15625                                 ; divide by 15625
     div    bx
     mov    bx,dx                                    ; Save the remainder
     mov    dx,ax                                    ; Cpy quotient to DX
     and    dx,00111111b                             ; Low 6 bits =
     mov    cx,dx                                    ;  remainder, save
     shr    ax,6                                     ; Divide by 64
     xchg   ax,cx                                    ; Get current rmnder
     mov    dx,15625                                 ; Multiply by 15625
     mul    dx
     add    ax,bx                                    ; Add in saved rmndr
     mov    bx,0                                     ;  to double-word
     adc    dx,bx                                    ;  remainder

; Now, the result is in CX and the remainder is in DX:AX.  Update
; the millisecond and nanosecond counts:

     add    word ptr DD_ReadData.Read_Millisecs,cx   ; Update millisecs
     adc    word ptr DD_ReadData.Read_Millisecs+2,bx ; BX is still 0
     mov    word ptr DD_ReadData.Read_Nanosecs,ax    ; Update nanosecs
     mov    word ptr DD_ReadData.Read_Nanosecs+2,dx

UpdateExit:
     pop          bx                                ; Restore BX
     ret                                            ; Near return to
caller
UpdateTimeStamp    ENDP

;*************************************************************************
;          Initialize - Device Driver Initialization routine
;       Called from Strategy to do device driver initialization.
;                     Discarded by OS/2 after use.
;*************************************************************************

Initialize         PROC

; Get and save the pointer to DevHlp
     mov       ax,word ptr es:|bx|.RPO_EndOfCode
     mov       word ptr DevHlp_Ptr,ax
     mov       ax,word ptr es:|bx|.RPO_EndOfData
     mov       word ptr DevHlp_Ptr+2,ax

; Initialize the 8253 Timer 0 to mode 2:
     cli                                  ; Clear interrupts for write
     mov       al,i8253CmdInitCtrZero     ; Set to mode 2
     out       i8253CtrlZeroOrTwo,al
     xor       ax,ax                      ; Write a zero initial count
     out       i8253CounterZero,al        ;  LSB, then
```

Figure 2. Source Code for the Timer Device Driver - TIMER.ASM (Continued)

```
      out       i8253CounterZero,al           ;  MSB
      sti                                     ; Reenable interrupts

; Use the 'SetTimer' DevHlp to chain our timer tick routine:

          mov       ax, offset Interrupt      ; Timer handler address
          mov    dl,DevHlp_SetTimer
          call   DevHlp_Ptr                  ; Call the SetTimer DevHlp
          jnc    NoError                     ; Carry clear = DevHlp worked

  ; DevHlp failed, cancel initialization:
          mov    es:|bx|.RPO_NumberUnits,0    ; Signal to cancel
          mov    es:|bx|.RPO_EndofCode,0      ; Zero code and data lengths
          mov    es:|bx|.RPO_EndofData,0      ; Zero code and data lengths

          mov   byte ptr es:|bx|.RP_ErrorCode,0ch         ; General Failure
          or    byte ptr es:|bx|.RP_Status,RP_StatusError ; Indicate error
          jmp   InitExit                     ; Get out

  ; DevHlp succeeded, normal exit:
NoError: mov    es:|bx|.RPO_EndofCode, offset Initialize
          mov   es:|bx|.RPO_EndofData, offset LastData

InitExit: or    byte ptr es:|bx|.RP_Status,RP_StatusDone
          ret

Initialize      ENDP
code            ENDS
END
```

Figure 2. Source Code for the Timer Device Driver - TIMER.ASM

determine how much to add to the running timestamp, and updates the running timestamp to return to the caller. Transferring this running timestamp value requires using the PhysToVirt and UnPhysToVirt device helpers to copy the running timestamp into the DosRead() caller's buffer correctly. As you can see, the routines used for the OPEN and CLOSE requests keep count of the number of users of the device driver.

The INTERRUPT routine first checks if the driver is being used. Whenever there are no active users of the device driver, the interrupt routine returns quickly without maintaining running time totals. This virtually eliminates any impact of having the device driver installed when it is not in use. If a user is active, the routine performs many of the same functions as the READ routine; that is, it gets the current tick count from the 8253 timer 0, uses the Last8253 tick count to determine how much to add to the running timestamp, updates the running timestamp, and sets Last8253 to the current tick count.

Notice the CLI and STI instructions in the OPEN, INTERRUPT, and READ routines. Interrupts are disabled at different times for one or more of the following reasons:

1. To read the 8253 ports.

To complete a read from the 8253, the driver issues a command (request counter latch) to a control register, then reads two one-byte registers in succession. If the driver did not disable interrupts, it might not be able to complete this sequence without some other code in the system affecting the 8253.

2. To prevent race conditions when updating shared variables.

Whenever the READ routine updates the running timestamp, or the OPEN or CLOSE routine updates the number of users, the driver must be careful that the INTERRUPT routine does not step in and read partially-updated variables. Disabling interrupts is a sure way to prevent this. Not only does it remove the possibility that the INTERRUPT routine gets control while OPEN or CLOSE is working, but also, because CLI locks out OS/2's scheduler, it prevents the OPEN and CLOSE routines from conflicting with each other.

3. To allow use of the PhysToVirt DevHlp.

The PhysToVirt DevHlp requires that interrupts be disabled while using a converted address.

Building the Timer Device Driver

Figure 3 shows a module definitions file, TIMER.DEF, used to link with the device driver. To build the device driver from this source code, use IBM MASM/2 or Microsoft® MASM and enter:

```
masm timer ;

link timer,timer.sys,,,timer.def
```

Installing the Timer Device Driver

Finally, to install the Timer Device Driver on your system:

1. Add the following to your CONFIG.SYS file:

   ```
   DEVICE=TIMER.SYS
   ```

2. Reboot your system so the changed CONFIG.SYS file executes

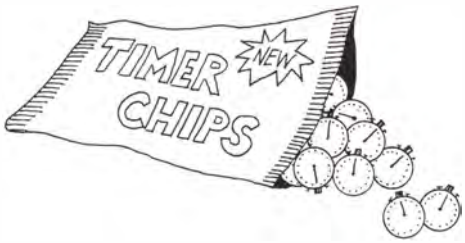
```
LIBRARY TIMER
PROTMODE
```

Figure 3. Module Definitions File TIMER.DEF

SUMMARY

The Timer Device Driver opens the door to high-resolution performance time measurement without added hardware or software expense. By assembling, linking, and installing the TDD, you can prepare any OS/2 workstation to perform high-resolution timing measurements. You then can add simple OS/2 calls to your programs to determine whether certain areas of your code perform as they should. The more you 'fine tune' the performance of your programs, the more pleased your customers will be.

Derek Williams, *IBM Corporation, 1001 W.T. Harris Blvd., Charlotte, NC 28257. Mr. Williams joined IBM in 1988, and is a Senior Associate Programmer in the Finance Services Industries Charlotte Development Lab. He received a BS degree in Information and Computer Science from Georgia Institute of Technology.*



Performance

One Stop Performance Shopping For The OS/2 EE Database Manager

by Bruce A. Tate and Tim J. Li

One of the delights of working with customers is discovering the variety of ways that they find to produce innovative enhancements and solutions for the OS/2® EE Database Manager. We hope that through this article, we can save you some time through these tips, many of which stem from real field experiences.

Most customers are happy with the performance of IBM®'s database engine once it has been properly tuned. Getting to the optimal state, though, can be tough. Difficulties that contribute to this problem include the complexity of the operating system, the wide range of performance improvement techniques, the variations in configuration and user applications, and the lack of performance tools and documentation.

We believe that a consolidated list of performance techniques is needed. Such a checklist, with general descriptions of tips, techniques, and suggestions, could be used by the OS/2 application developer as a shopping list for the performance aspects of database application design.

DATABASE DESIGN

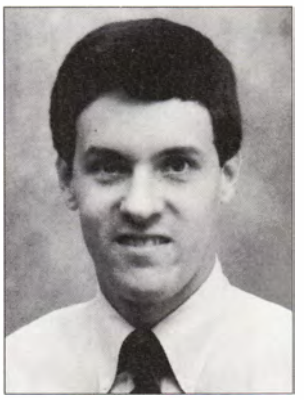
After the functions and services of an application are well understood, the next step in application design often is the definition of the data structure and relationships among the data elements. For a database application, this is the database design stage. Index design and database partitioning should also be considered at this stage of design. Many of these decisions will impact system performance, as well.

Table Design

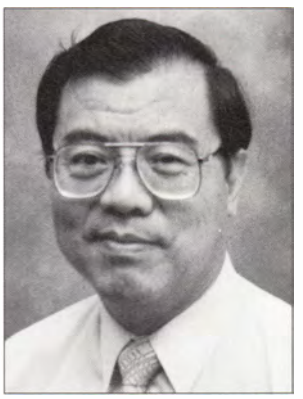
Normalization techniques are beyond the scope of this article, but the idea here is to minimize joins where appropriate. Though separating tables may be a good technique to reduce redundancy and thus improve capacity requirements, sometimes it is worthwhile to merge these tables into one. If it is known that multiple tables will be joined frequently and that the tables could be considered logically as a single table, then merging the tables could save many costly joins in the future. For example, consider two tables: person and address. It may make sense to separate these tables to reduce redundancy if more than one person may live at the same address. This would avoid having to enter the same information several times. However, if most reports require information from both person and address (for example, home city and state, and person's social security and employee numbers), then it may be better to merge these tables into one. The tradeoff, then, is performance versus redundancy and capacity. Tables should be normalized completely first, and then recombined where needed on a case-by-case basis.

Index Structure

The importance of indices cannot be over-emphasized. In read-only situations, it usually is worthwhile to create an index on fields which will be used for joins, or otherwise will appear in 'WHERE' or 'ORDER BY' clauses of an SQL query. Indices should always exist on primary keys of larger tables. For a multiple column index, the most restrictive column should be listed first. In some situations, table access may not be required at all if all of the fields needed by a query exist in the index.



Bruce A. Tate



Tim J. Li

Here's a performance shopping list for database application developers

For example, consider table T and an index I in the form:

```
Table T = (A, B, C)
Index I = (A, B).
```

For the query:

```
SELECT A,B FROM T WHERE B = 5,
```

the database manager only need access the index, and not the table itself. Creating appropriate indices sometimes can improve response times by 90 percent or more!

There are, however, situations where creating indices can degrade system performance. Updating, deleting from, or adding to a table with indices may take longer than changing a table without indices once the record is found, because the indices also must be updated. If a table is write-only or has more writes than reads, then this may be a serious consideration. Most tables, however, do not fit this description.

Similar ideas can be applied when populating large tables with imports or inserts. In general, it is faster to import a table and then create the index than to create the index and then import the table. This is true because creating an index from an existing table is faster than building an index one row at a time during an import.

The 'EXPLAIN' tool can be used to see if an index is used in a particular query. Through intelligent use of this tool and good table design, significant performance improvements may be realized. EXPLAIN is available selectively by special request from IBM.

Single Databases

It is important to limit the number of databases on a server carefully. Significant resources are allocated whenever multiple databases are opened on a single server. Several control blocks and data areas are allocated by the Database Manager for each active database, such as the buffer pool and sort heap. Often, a better strategy is to place all of the tables from multiple databases within a single database. Naming conventions, the Grant-Revoke function, and qualifiers may be used to keep the tables logically separate.

APPLICATION DESIGN

When the database design is complete, then the application really starts to take shape. Here, many decisions must be made that will have significant impact on the final form of the application. Further, many of these decisions may be difficult or impossible to change later in the process. For example, "Should 'COMMITS' be issued for each update?" Or "Should Application Remote Interface (ARI) be used for this application?" It is important to consider performance when making these decisions.

Locking

When considering the performance implications of a locking strategy, the programmer must balance the conflicting goals of concurrency and data integrity. A reading process and a writing process may not hold a lock on the same object. Processes may hold locks to read the same object. No two processes may hold locks to write the same object. Obviously, if some process has a table write-locked, then no one else can lock the table until that process commits or rolls back. As a general rule, then, in a multi-user system, it is better to lock records than tables. This is the default locking mechanism. In other situations, such as single user systems or applications which have private tables, it may be desirable to force an exclusive lock on a table to prevent further locking overhead. The statement to accomplish this is "LOCK TABLE **<TABLENAME>** IN EXCLUSIVE MODE".

Other situations can cause table locks, such as a table scan when the isolation level is "REPEATABLE READ". Also, if the "LOCK LIST PAGES" configuration parameter is too low, then the DBM will run out of memory for storing row locks, and several row locks may be traded in for a single table lock. This is called lock escalation, and is managed internally by the database manager. It can have significant impacts on performance. Generally, for performance in multiuser systems, it is better to avoid table locking whenever possible.

COMMITS

Managing commits also involves two conflicting goals. Commits do take a certain amount of overhead, in the form of log and buffer pool maintenance. On the other hand, commits free resources that may be used by other processes, most notably locks and logs. It is important to understand that locks are accumulated on the system tables even in read only SQL (selects), and even in the "UNCOMMITTED READ" level of isolation. Therefore, it is important to commit even in these situations to free up locks that the DBM has placed on system tables. Following are some general rules to keep in mind when formulating a "COMMIT" strategy:

1) For data integrity, never commit in the middle of a logical unit of work.

2) Increase the number of COMMITs to improve concurrency.

3) Reduce the number of COMMITs to improve the performance within a single process when the concurrency impact may be tolerated.

4) Always commit, even for applications with the "UNCOMMITTED READ" level of isolation, and also for "SELECT ONLY" transactions.

Application Remote Interface (ARI)

ARI is one of the most important, yet least used, of all performance techniques. It is used on the requester to trigger the execution a block of code on the server. This block of code, often referred to as a remote procedure, is stored in an ordinary OS/2 DLL (dynamic link library) or REXX routine. Practically any block of code can be executed in this way. The remote procedure may or may not invoke the database manager. The best strategy, though, is to group together several SQL calls into a single procedure and execute this procedure through ARI. Combining several different queries into a single procedure to be executed on the server improves system performance in several ways. First, communication costs between the server and requester may be reduced drastically. Second, the faster microprocessor on the server may be utilized. Instructions that normally would be executed on the requester are moved to the server, which typically is a faster machine. Finally, the intermediate control blocks required by Remote Data Services for each SQL call do not need to be shipped for every call. This leads to further savings of the requester CPU, and can offset the additional responsibility of the server CPU.

To use the ARI, precompile, compile, and link the remote procedure to be called into a DLL. Place the DLL on the server, and use the database manager API to call the remote procedure. Data may be passed to and from the remote procedure using SQLDA's. The potential benefits of ARI were shown recently in NSTL tests.[1] Figure 1 shows the performance of ISBN Order in NSTL book order benchmark with and without ARI.

Static vs. Dynamic SQL

The Database Manager supports two different types of SQL: static and dynamic. Static SQL is bound before the program is executed, either at precompile time or bind time. With static SQL, the statements to be run must be known in advance. Dynamic SQL is *bound* at run time. It must be bound each time the cursor for the statement is opened. Obviously, it is better to bind only once when the '.exe' file is created instead of each time that the program is executed. Therefore, when the programmer knows in advance which SQL statements will be run, it usually is better to use static SQL.

An exception is an SQL statement which contains the fragment "LIKE :variable". Here, the optimizer does not have enough information to decide if an index is usable. If the "LIKE" clause looks like "ABC%", then an index may be desirable, and could be used for dynamic SQL. It is quite possible that this query could run faster with dynamic SQL than static SQL, since it is able to take advantage of an index!

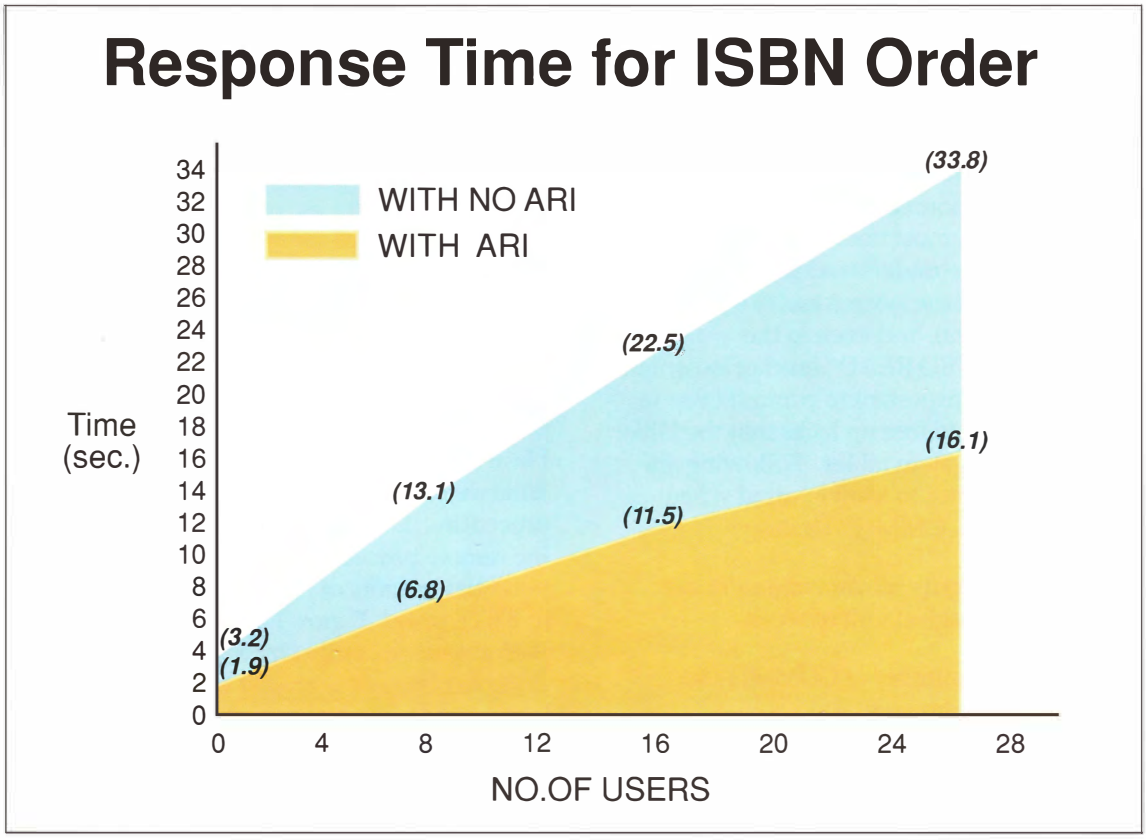


Figure 1. Response Time for ISBN Order With/Without ARI.

PS/2 HARDWARE CONFIGURATION

One popular suggestion for improving performance is to buy newer, more expensive hardware. Since this is not always feasible economically, then performance trade-offs must be understood. By understanding database performance requirements and trade-offs, the user can make better decisions about system configuration.

Splitting the Log and Database

Better performance can be achieved by placing the database and the database logs on different physical hard disks. This is true because some degree of concurrency can be realized by accessing both drives at the same time. SCSI devices can make the improvements even more pronounced. Also, the disk head movement may be reduced significantly. If only one physical drive is available, then it is better to place the logs and the database on contiguous partitions or even on the same partition. Also, the partition(s) should be as close as possible to the center of the drive. For example, if a hard disk was partitioned into drives C,D,E, and F, a better configuration would be to place the database on drive D: and the log on drive E:. This minimizes lateral movement of the disk head.

Memory

If there is not enough memory on the server to handle the number of workstations adequately, then performance will degenerate drastically as activity and the number of database connections increase. Each remote application which issues a "START USING DATABASE" (each connect to a remote database) will use an additional 120-150Kb of memory at the database server. When the amount of working memory exceeds the amount of installed memory, system performance can degrade seriously due to swapping

Processing Power

Database manager applications in a heavily loaded system usually are CPU-bound . Therefore, improvements to the CPU may

have significant positive effects on transaction throughput or response time. There are exceptions, such as applications with low concurrency, or high update activity. In these cases, the performance gains from a faster CPU may not be as pronounced.

PERFORMANCE RELATED FEATURES

Database Manager provides a series of features, utilities, and functions that may be used to improve system performance. They come in the form of functions that organize database data, precompiler switches that improve the blocking of data, and similar switches that allow performance vs data integrity tradeoffs. Most of the battle can be won by knowing that these features exist, and when to take advantage of them.

Record Blocking

Record blocking can improve the performance of read-only queries which read multiple database rows. For such queries, when the requester does a single fetch, a block of data is moved to the requester. The requester then reads from its private block instead of requesting additional single rows from the server. Record blocking is transparent to the user application. Record blocking is a bind time parameter. To ensure that record blocking is used only where it can be effective, use the '/K=UNAMBIG' option when binding, and specify cursors which will not be updated, deleted, or inserted as 'FOR FETCH ONLY'. This series of steps will ensure record blocking for all fetch transactions. The default is 'UNAMBIG', which uses record blocking for all cursors which are fetch-only. Record blocking is a way to improve performance significantly without doing much work. Gains will be most pronounced when many database rows are expected to be returned from frequently used queries. If record blocking is desirable, to ensure that record blocking will be used for dynamic SQL the record blocking parameter should be set to 'ALL' at bind time .

Isolation Levels

Another bind time parameter is the isolation level. Isolation levels let users trade additional concurrency for data integrity. From strongest to weakest in data integrity, they are:

Technique	Implementation Time	Expected Payoff	Development Cost
Turn Trace Off	Any	High	None
Increase Memory	Any	High	Low
Index Restructure	Any	High	Low
App Remote Interface	Early Development	High	High
Record Blocking	Any	High	Low
Commits	Development	High	Medium
Table Restructure	Early Development	High	High
Query Mgr Row Pool	Any	High	Low
Static SQL	Development	Medium	Medium
Reorg/Runstats	In Production	Medium	Low
Split Log/2 Drives	Any	Medium	Low
SQLLOO	Any	Medium	Low
HPFS/FAT partitions	Any	Medium	Low
Locking	Development	Medium	Medium
Isolation Levels	Late Development	Medium	Low

Figure 2. Performance Techniques

Repeatable Read, Cursor Stability, and Uncommitted Read. From fastest to slowest, due to concurrency, they are Uncommitted Read, Cursor Stability, and Repeatable Read. If the isolation level required is not understood clearly, then it is best to err on the side of caution and use the strongest. Otherwise, it is best to use the weakest (and thus fastest), isolation level that can be tolerated by the application. Usually, this turns out to be "CURSOR STABILITY". It is important to note that an uncommitted read could let the application fetch a value that was never committed, and thus never really existed in the database!

In general, the Repeatable Read never will yield inconsistent results. Cursor Stability usually is acceptable unless there are dependencies among several rows of the same table that must be preserved. Uncommitted Read only is acceptable in situations which do not require consistent data, or in situations where it is known that the data being examined will not change.

SQLLOO

SQLLOO (SQL LAN Only Option) is a faster communication protocol than APPC. However, SQLLOO is not always a viable alternative. SQLLOO is the default communication protocol for Remote Data Services (RDS).

REORG/RUNSTATS

After significant insert, delete, and update activity, a database table may become fragmented significantly, and the statistics used by the optimizer may become outdated, forcing bad optimization strategies and overall performance degradation. Running statistics can update the optimizer statistics for a table. Reorganizing the table resolves the fragmentation problems. It also puts the database records in the order of a specified index, which may improve performance significantly. The benefits of a 'Reorganize' will not be realized unless statistics are run afterwards. Running statistics, however, does not mandate a reorganization. Similarly, after a RUNSTATS, a rebind is necessary.

METHODOLOGY FOR PERFORMANCE EVALUATION

There is a wealth of strategies for benchmarking and tuning. These suggestions have been proven to be successful in our Austin lab, where we frequently tune systems and measure overall system performance.

Performance Management Hooks

In order to be able to create the system with the best performance possible, it is important to have the best performance data possible. A good way to gather these data is from trace points placed within an application. Each trace point should record a unique identifier and the time that the trace point was encountered before and after the function being measured. Other data, such as the process ID, also may be recorded. In order to get the most information without impacting system performance, this information should not be written to a file on the database server. Possibilities for storing the collected data include memory buffers on the requester or server (if memory is not a bottleneck), or files on the requester (if there are enough requesters to keep the database server busy). It is important to save data from old runs, so that an accurate picture of an application's performance history may be assimilated and the performance characteristics of all components of the current system assessed over time.

Prototyping Simulation

Benchmarking may be used to gain information that will help predict system performance. A good benchmark will represent the real world as closely as possible in as many areas as possible, including the number of users, size and composition of database data, the type and order of requests, and the LAN topology and system configurations. Notice that the prototype does not have to look like the final database application, it merely has to act like it. For this reason, many people use a script of some type. Care should be taken, though, to consider the amount of conflict that is to be expected in the real world. For example, a script containing

updates to the same record on all machines would generate locking conflict that may not be expected in the real world. Similarly, if none of the data sets overlapped, the amount of conflict could be artificially low. The buffer pool also comes into play. If the same small data set is read over and over, then the DBM is reading from an in-memory buffer instead of from disk, which could skew results significantly .

Benchmarking and Tuning Methodology

When tuning the Database Manager for performance, it is important to control the entire process tightly. There are several things to keep in mind. First, several performance runs must be made. The initial run should be discarded, as DLL's must be swapped in, and the DBM initialized. Afterwards, several runs should be made under identical conditions and the average of all runs taken. Second, only a single parameter or condition should be changed between performance runs. This is the only way that the results of each benchmark can be tracked and assessed accurately. Finally, the parameters for which the largest gain is expected should be changed first. In this way, the system may be tuned until some goal is reached, and the tuning process completed with the confidence that the most significant parameters have been refined.

OS/2 EE CONFIGURATION

The Database Manager has a wealth of tuning parameters and switches. Some are highly specialized, and will not impact overall performance much. Here, an effort was made to hit the highlights to let tuners concentrate on those parameters which are most likely to help system performance.

TURN TRACE OFF

The OS/2 trace can slow down the system, sometimes by a factor of two or three. If performance degenerates unexpectedly, first make sure that the OS/2 trace has not been turned on inadvertently.

HPFS vs. FAT

OS/2 has an option for a new High Performance File System (HPFS). It has features such as a cache and better file organization. The Database Manager cannot take full advantage of all features of HPFS. This is particularly true of the cache. Since data reliability is so important to the Database Manager, all log records must be written to the disk immediately. This defeats the purpose of the cache, which tries to avoid disk activity by keeping frequently accessed information in memory. The database manager implements its own cache, called the buffer pool. Even so, the database is usually faster on HPFS. The log is slightly faster on FAT, the other file system for OS/2.

Query Manager Row Pool

Query Manager scrolling performance can be improved by increasing the size of the Query Manager row pool. Beginning with release 1.2, Query Manager began formatting an entire row of data instead of just a screen of data. This made horizontal scrolling smoother, but required a larger row pool. For this reason, the default row pool is no longer adequate for many applications. Note that this impacts the scrolling performance in Query Manager only. This is a Query Manager startup parameter.

Database Manager Configuration Parameters

The Database Manager configuration menu may be accessed from within Query Manager. It contains parameters which are used by the DBM to manage resources which are allocated once, when the DBM is started.

Shared segments: 802

Parameter	Expected Payoff	Suggested Value	Priority
Shared Segments	Low	802	1
Databases	Low	(# databases)	2
Req I/O block size	Low	default	3
Srv I/O block size	Low	default	3
Remote Connection	Low	# remote cn	2

Figure 3. Database Manager Configuration Parameters

A heap is a large block of memory from which smaller blocks are taken. Since memory is allocated dynamically as this heap grows, for a database server we recommend to set this parameter at the maximum value of 802.

Figure 3 shows several other DBM configuration parameters. However, these do not affect system performance significantly, and we recommend they be left at their default values.

Database Configuration Parameters

The Database configuration menu also may be accessed from within Query Manager. It contains parameters which are used by the DBM to manage resources which are allocated separately for each database that is started. Here are some suggested database configuration parameters for a database server.

```
buffer pages >= 250  (server only)
```

This parameter is probably the most important parameter to consider when tuning database performance. If it is not large enough, performance will be degraded significantly. If the parameter is much too large, then additional system resources will be used (memory). It is more important to make the parameter large enough than to try to minimize it. (See Figure 4.)

Log File Size	Medium	1000	1
Primary Logs		3	1
Secondary Logs		2	1
Soft Check	Low	default	3
Buffer Pages	High	250	1
Deadlock Check Int	Low	default	3
Lock List Pages	Medium	25	1
Lock List % per App	Low	100/# apps	1
Max Open Files	Low	Default	3
Max Appl Open Files	Low	Default	3
Maximum Applications	Medium	Number Cons	1
Application Heap	None	Default	3
Communications Heap	None	see text	3
Sort List Heap	High	3	1

Figure 4. Database Configuration Parameters

```
Lock List Pages = 25 (4Kb pages)
```

This parameter will impact concurrency if it is too small. If the application is expected to hold many locks concurrently (e.g., long transactions with Repeatable Read level of isolation; long transactions that update, insert, or delete large numbers of records), then this parameter must be increased. Increase if concurrency problems show up, such as too many rollbacks due to deadlock.

```
Lock List Percent per Application
= 100 / number of applications
```

The "LOCK LIST" tells the system how large to make the overall "pie". "Lock list percent per application" describes the largest piece of that pie that any one application is allowed to accumulate.

```
deadlock check interval =
(default)
```

This parameter does not impact system performance significantly if the system is behaving as it should. If there is artificially high contention in the system, either because commits are not happening frequently enough or because the "LOCK LIST" is set too low, then deadlock checking could be a problem. In this instance, though, it is better to solve the problem which is causing the high contention: by committing more frequently or by increasing the "LOCK LIST", or both.

```
Sort List Heap = 2 (64Kb Segments)
```

This should be increased if large tables are sorted or joined. This also is an important parameter to tune.

Suggestions for Log Configurations

A small number of large logs is better than a large number of small logs. Logs in OS/2 1.2 are circular in structure. The same log space is used over and over. When a piece of a log is no longer needed, the space is reclaimed. In this process, secondary logs are reduced to a file of size zero and must be reallocated if needed again. At this point, users will see a noticeable delay. The bigger the log space, the longer the delay. The primary logs, then, should be large enough to handle most normal situations, since they are allocated only once when the database is used by the first application for the first time.

To tell how much log space is required, some exploring is needed. A good procedure is first to note the log path from the appropriate Query Manager database configuration screen. Then, set the primary logs to 0 and the secondary logs to some high number. The database should then be rebooted, and the application in question run until the log stops growing. At this point, a simple OS/2 directory can be run on the logs:

```
DIR C:\SQL00001\*.LOG
```

where C:\SQL00001 is the current path specified in the log configuration. The timestamp will have changed for all secondary logs which were allocated. With this information, the appropriate number of primary logs may then be allocated. As discussed previously, for improved performance it is better to have the logs on a different physical hard disk from the database. These suggestions may be combined for a superior log configuration.

We encourage you to use these techniques as performance shopping lists, so that your application may use the rich performance functionality that has been provided by the OS/2 EE Database Manager.

ACKNOWLEDGMENTS

Many thanks to Laura Camp, Tim Malkemus, Bob Russell, and Phil Welti for their friendship and willingness to share.

REFERENCES

1. **IBM OS/2 Extended Edition with ARI** *Software Digest Buyer's Alert*, Vol. 7, No. 13.

Bruce A. Tate, *IBM Personal Systems Programming Center, 11400 Burnett Road, Austin, TX, 78758. Mr. Tate, a Senior Associate Programmer, joined the Austin Database team in 1987 where he worked as a tester and programmer on the OS/2 EE Query Manager project. In 1990, he began his current assignment, where he forms close customer relationships to provide education and technical support for OS/2 application developers and customers. Mr. Tate received a BS in Computer Science and Math from Mississippi State University, and currently is pursuing an MS in Computer Science at the University of Texas at Austin.*

Timothy J. Li, *IBM Personal Systems Programming Center, 11400 Burnett Road, Austin, TX, 78758. Dr. Li is a senior programmer focusing on OS/2 EE Database Manager performance analysis. He joined IBM in 1969 as an engineer in the area of system reliability in the System Development Division. He has since been involved with numerous assignments at IBM, including performance analysis for communications systems, 5520, office systems and strategic planning for OS/2 EE Communications Manager. He holds a BS degree in Electrical Engineering from National Taiwan University and Ph.D degree in Electrical Engineering from Purdue University.*

Software Tools

OS/2 Freeware and Shareware

Brian Proffit

by Brian Proffit

Not all development tools come in shrink-wrapped packages. This month's Tools Update looks at other sources of tools and utilities.

In this issue, we leave the glamor world of big name companies and products. We will instead take a to look at other useful programs for the developer that may not cost a dime! Out of its early *hacker* days, the bulletin board service (BBS) community has grown into a phenomenal force in the industry. Those services act as a *user group* where people can compare notes, help each other solve problems, and share programs that they find interesting or fun. This grass roots network has contributed heavily to the success (and failure) of many personal computing products.

This is as true for OS/2® as it is for other operating environments, and perhaps more so. The development community appreciated the power of OS/2 long before the users. The first thing those developers went to work on was the conversion of their favorite tools and utilities from bulletin boards. The result is that there are literally thousands of files available for OS/2, from analog clock displays to sophisticated editors.

WHAT IS SHAREWARE

First, let's understand the terminology. At one end of the scale are commercial products like IBM's Programming Tools and Information. These can only be obtained by purchasing them. The other end of the scale is *freeware* or public domain software. The authors of these programs have made them freely available for distribution. This category defines the vast majority of programs found on bulletin boards.

At these prices there is usually no support, and the documentation (if it exists) is often a README file. These shortcomings do not mean that there aren't good packages. There are many excellent programs to be found.

Between these two extremes lies a new phenomenon known as *shareware.* Many small software companies, finding it difficult to establish a distribution channel for their product, decided to take advantage of the BBS community. These packages may be freely distributed, and there is no charge to download the program. The user is given license to use the program for a limited period of time, typically thirty days. At the end of this period it is expected that the user will either delete the program from their computer, or register their ownership and send a check to the company that developed the code. These are usually more professional looking packages and often come with extensive documentation and/or on-line help. There is often support available to registered customers.

In order to provide additional user security, the Association for Shareware Professionals was formed. This association of companies work to help demonstrate to the users that these are indeed professional software packages. There is an Ombudsman's office that will take user complaints toward products or companies represented by the Association, and work with the company to resolve the complaint.

SHAREWARE FOR OS/2

One of the most popular bulletin boards devoted to OS/2 is the OS/2 Shareware BBS run by Pete Norloff in Fairfax, Virginia. Norloff is a programmer. A couple of years ago his company had a need to move from DOS to a multitasking platform. The company selected OS/2 over UNIX®, so Pete began his OS/2 education — in large part via bulletin boards. "I was haunting the electronic ether for a long time," Norloff says. He gathered loads of valuable OS/2 information as well as quite a collection of programs. In March 1990, he decided to share all he'd gathered with the rest of the world. "I had a spare PC and phone line, got an inexpensive modem, and it took off from there!"

He now has three phone lines running into a 25 MHz 386 running OS/2 SE 1.3. He's adding a second 150MB disk drive to keep up with the demand. Just how much interest is there in an OS/2 bulletin board? "I've been getting about 140 calls per day," Norloff reports. His membership comes from 47 of the 50 states, 6 Canadian provinces, and 14 other countries. "I've got one guy who calls in from Sweden regularly at 1200 bps! If people are willing to do that, there's obviously a need that I'm filling." The BBS now has over 1700 files available for downloading, all at no charge. To try it out at speeds up to 2400 bps, call 703-385-0931. For speeds up to 14,400 bps, call 703-385-4325.

FINDING A BBS NEAR YOU

While the OS/2 Shareware BBS is one of the largest, it certainly isn't the only bulletin board specializing in OS/2 users. In fact, Evergreen Systems, in Marlton, New Jersey, has started an OS/2 BBS Locator Service. Just call 609-596-1267 and a voice system will tell you if any OS/2-specific bulletin boards are listed in your area. If not, it then lists bulletin boards not in your exchange, but in your area code. There's also a function that allows sysops of OS/2 boards to leave a message to get their board added to the database.

Evergreen specializes in interactive voice response systems like voice mail and automated teller functions for banks. They use OS/2 Extended Edition in their client systems. Being familiar with OS/2 boards, Evergreen's head, Gary Green saw helping others find a local BBS — a natural addition to his company's services. This also gave Evergreen a chance to promote other services.

"We're expanding the service to include a list of OS/2 retailers," Green said. Since not all dealers will always have a full supply of OS/2 products, the Evergreen system will allow people to dial in and get a listing of dealers in their area. In fact, they're instituting a fax-back service for those who want to receive a printed list rather than one by voice. The user can call using a fax machine, select the option from the voice menu for fax service, then switch their fax machine to print mode and the list will be sent in a fax bitmap form.

IBM SUPPORT FOR BULLETIN BOARDS

IBM's National Support Center in Atlanta runs a BBS that is open to all. This BBS also offers sections for discussion of PS/2 hardware, applications, and nearly anything else that is related to OS/2. They can be reached at (404) 835-6600.

The grandfather of bulletin boards is the CompuServe Information Service (CIS). It started as a news and information service, it is the messaging and library areas that have contributed most heavily to CompuServe's growth to around three quarters of a million subscribers. The ability to reach so many people in one place was what attracted IBM to CIS. The IBMOS2 forum was opened on CIS June 4. In its first two months the forum gained over 5000 members, received over 15,000 messages, over 300 files for downloading, and had guest appearances by IBM's Asst. General Manager for Programming, Personal Systems, Lee Reiswig (aka The Blue Ninja).

THINGS TO LOOK FOR

So, once you've found a bulletin board, what kinds of tools are available for download? Here are some interesting files I've found:

CVTICO helps you migrate your applications to OS/2 by converting your icons from Windows to OS/2 format. CVBMP performs the same function for bitmaps.

FSHL21 is an enhanced command line shell for OS/2. It provides aliasing to streamline OS/2 usage for those who prefer the command line interface. Another handy utility for those using the command line interface is NPIPE. This filter allows the user to redirect STDOUT to a named pipe so it can be read into another window.

To manipulate information in the clipboard, try CLIPEDIT. It will allow you to view, alter, print, or save the clipboard contents.

If you spend much time looking at binary data, ANA may boost your productivity. This binary analyzer applies some intelligence to its display, identifying strings and displaying them in ASCII form, etc.

Do you need a variety of different push-button combinations in your dialog box? BTNPLC can prevent the work of creating several different dialog boxes. Create a generic DLG then pass it the push-button information in an array. BTNPLC will automatically arrange the array at the bottom of the dialog window.

PMTEST is a record/playback utility to help automate program testing.

There are several UNIX-like utilities available for OS/2, such as GREP, EMACSOS2, OS2-YA (YACC), AWF, SED, and DIFF.

There are also some valuable tips and techniques files, such as DOSSYS which describes the interface to the DOS.SYS driver. This allows a Presentation Manager™ program to spawn a session in the DOS box automatically. TIPF is a VIEW-based program which displays a sizable list of OS/2 applications and tools, including company and contact information.

Finally, if you've got a hybrid system with a Brand X disk drive and a Brand Y display adapter, there is an impressive collection of OS/2 device drivers available in the BBS world.

MORE THAN JUST DOWNLOADS

Those who use these on-line services just for downloading software are missing out on an important part of the BBS world. There are thousands of people out there who have been developing for OS/2 and are willing to help others with questions and problems. Our team for the CompuServe® IBMOS2 forum is largely a volunteer effort that takes on questions on virtually anything related to OS/2. One of the most interesting aspects, though, is the number of times users help each other! Spend some time in the message areas, and you'll find many people just like yourself who have already solved the problem you're facing.

Considering the hundreds of packages there are available in the BBS world, it is a virtual certainty that there is something of value there for everyone. Take the plunge into *cyberspace*, but remember: the key rule of these boards is the spirit of giving. If you use a shareware tool, contribute to its author. If you develop a useful tool or utility, why not share it with the rest of the world?

Happy computing.

Brian Proffit, *IBM Entry Systems Division, 1000 NW 51st Street, Boca Raton, FL, 33429.* *Mr. Proffit is a senior programmer in OS/2 software developer strategy. He is currently responsible for OS/2 developer tools. He joined the development laboratory in Boca Raton in 1983 after working as a systems engineer in Dallas.*

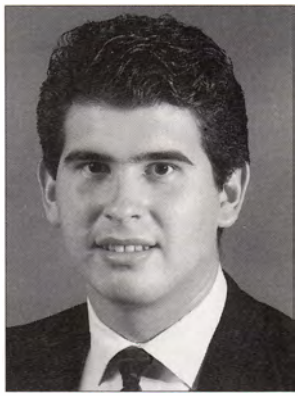
Software Tools

The Arcadia CUA Workbench: A GUI-Independent Tool for the Creation of Robust OLTP Front-ends

by David C. Pacheco

The Arcadia CUA Workbench™ provides a set of GUI-independent tools for the creation of robust, high performance, easy to maintain front-ends for Online Transaction Processing (OLTP) and client-server environments. The Arcadia CUA Workbench consists of three components:

- The Application Generator™ is an interactive What-You-See-Is-What-You-Get (WYSIWYG) screen painter and rapid-prototyping environment. The Application Generator (AG), which runs on OS/2®, lets you animate an application and test-drive it. The AG creates a script file which describes the structure and properties of graphical objects using a tag markup language based on the ISO Standard Generalized Mark-up Language (SGML).[1] You can think of it as the PostScript® of GUIs. Advanced users can directly edit and update the SGML tag files and add new graphic objects. Arcadia will keep enhancing the SGML language to keep up with new Presentation Manager™ and Windows® constructs.
- The Script Compiler transforms the GUI-independent objects into GUI-specific ones (see Figure 1). Arcadia provides a compiler for DOS and Windows 3.0 and uses a Tag Language compiler for OS/2 PM provided by IBM®. Arcadia plans to provide its own OS/2 PM compiler as well as compilers for other GUI environments such as Macintosh®, Motif®, and OpenLook®.
- The Runtime Interface provides twelve SAA-based API calls for interfacing with the compiled GUI-specific objects at runtime.[2] Arcadia provides interfaces to C, C++, Pascal, Cobol, and Fortran. The API calls allow the manipulation of the graphical objects from within your programs and the sharing of variables (see Figure 2).



David Pacheco

DESIGNED FOR CLIENT-SERVER OLTP APPLICATIONS

Development tools for OLTP client-server applications must be architected differently

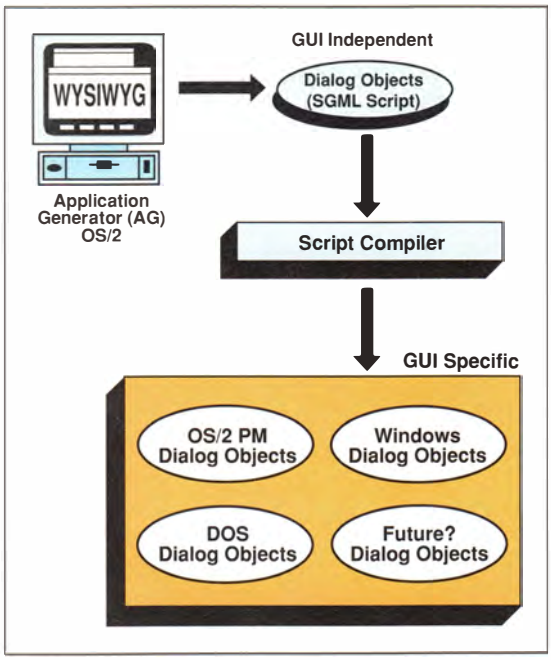


Figure 1. The Arcadia Build-time Environment

The AG creates a script file which describes the structure and properties of graphical objects

The Arcadia CUA Workbench is a multi-platform, OLTP application generator

from other GUI tools. Arcadia's Application Generator is designed for this environment.

Developing OLTP in a client-server setting requires intensive programmer involvement.[3] Programmers write the transaction code on the server, work out the semantics of the network exchanges, and develop the client application. An OLTP GUI client application must be able to handle both user-in-control and server-in-control situations. It must also handle the issuing of remote procedure calls and accepting of commands and data requests from the server. To do all that, it is best to have a procedural, third-generation language in control at the client with the capability to issue calls to the GUI environment when display services are needed. To execute commands from the server, the semantics of calls must be at the level of display panel or add pop-up.

In addition, the programmer should be able to associate predefined graphical objects such as fields, forms, menus, list boxes, and push buttons with programs. These associations are event-driven, so event handlers must be coded to provide the logic of the application. This approach is different from that of decision-support oriented tools, which provide canned event handlers.

The unique API approach of the Arcadia CUA Workbench, and the power of the WYSIWYG tool to create event-driven intelligent objects, allow a clean separation of the code from the user interface. These features make Arcadia CUA Workbench the ideal tool to fulfill the dual needs of OLTP Client-Server applications. The following describes in more detail the five key requirements of GUI development for client-server OLTP applications and how the Workbench satisfies them.

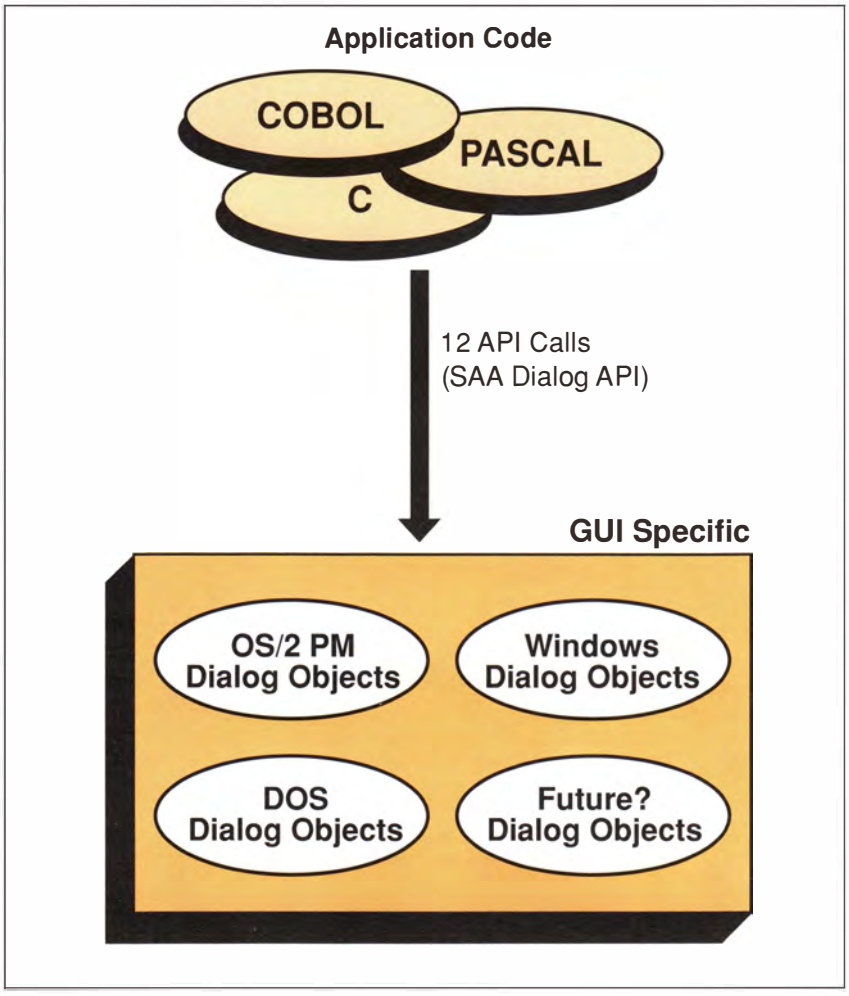


Figure 2. The Arcadia Runtime Environment

1. 3GL SUPPORT FOR ROBUST CLIENTS. The Arcadia CUA Workbench does not require you to learn another language for the non-GUI aspects of the client code, but allows you to use a standard 3GL. The dialog objects created with the AG can be manipulated directly by applications written in C, Cobol, Pascal, and other standard 3GLs.

Event-handling procedures are written directly using a 3GL. Developers use the language that best suits their needs. There is no need to learn a proprietary language to create graphical applications. This greatly reduces the learning curve while allowing developers to work with their favorite compiling and debugging environments.

2. SIMPLE DIALOG API. Programs interact with dialog objects using a simple application programming interface (API) consisting of 12 verbs. They implement the SAA™ Common Programming Interface™ dialog services.[2] These verbs include facilities to display screens and dialog windows, as well as pass variables between the dialog objects and your application (see Figure 2). The same API is used for all GUI platforms.

Thus, the application manipulates visual objects at a logical level, without having to get involved in the inner workings of the GUI. There is less code to write and maintain because all of the GUI objects are built and maintained separately, using the AG's WYSIWYG facilities. Developers and support people never have to learn complex GUI languages, such as OS/2 PM or Windows 3.0. This is especially important when developing applications to work on multiple GUI platforms.

3. CLIENT OR SERVER APPLICATIONS CAN BE IN CONTROL. You can design your client application to allow either the user or the server to be in control, or both. You can use the WYSIWYG to associate 3GL event handlers with appropriate visual objects, such as push-buttons and menus, to put the user in control. Additionally, the Workbench's unique architecture allows each application to use the 12 command API to manipulate the user interface and display visual objects, including data. Through this mechanism, the client or the server can dispatch instructions to the GUI. For example, a workflow server can tell the client program which screen to display next. You can tailor the application to its requirements, not the tool's limitations.

4. COMPILED FOR PERFORMANCE. Usually the price for such flexibility is an interpreted script-oriented environment. But interpreted applications do not meet the stringent performance requirements of OLTP front-ends, which often require users to perform highly repetitive tasks quickly, hundreds of times a day. The Arcadia CUA Workbench solves this problem in three ways. First, your client application is compiled because it is written in a 3GL. Secondly, the Script Compiler transforms the GUI-independent scripts from the WYSIWYG editor into the targeted, compiled native GUI code, such as Windows 3.0 or Presentation Manager. Finally, the Arcadia run-time environment maintains a cache in memory of the most recently used panels. The caching and compilation features work together to provide the performance OLTP applications require.

5. PANEL-ORIENTED DESIGN. The AG's WYSIWYG facilities create and track panels. A panel is a container of GUI objects, such as fields, buttons, and help text. Through the WYSIWYG, you create, edit, debug, and display panels. At runtime, your application displays and navigates through these panels. Many layers of pop-ups can be placed on top of existing panels through the API. The caching algorithm described above works with this concept of panels.

OLTP applications naturally work well with this panel-oriented paradigm because the panel mirrors a business form and is a suitable entry mechanism for business transactions. This simplifies application design, programming, and maintenance.

POWERFUL CUA-COMPATIBLE SCREEN GENERATOR

The Application Generator (AG) allows you to create visual objects, or screens, through an interactive WYSIWYG environment (see Figure 3). These objects include pushbuttons, scroll bars, list fields, help, message boxes, and others.

Double clicking on any object allows its properties to show up in a dialog box

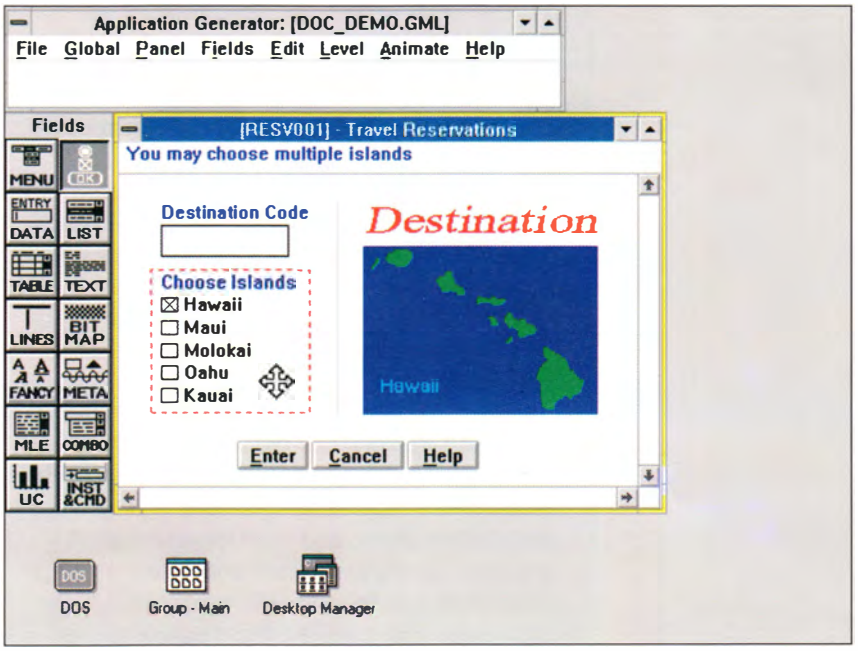


Figure 3. The Arcadia Runtime Environment

Without writing a line of code, you can define and associate data validation rules with each field

WYSIWYG Environment

Through mouse drag-and-drop interaction with the AG, you create a wide variety of graphical screen objects and define their attributes and object behavior rules. Double clicking on any object allows its properties to show up in a dialog box. For example, double clicking on a field could produce the dialog box listing the attributes associated with it. The toolbox for defining objects can be accessed either through the menu bar, or through a movable (and hideable) icon palette. This very complete environment allows you to create intelligent, self-contained GUI forms rapidly and without writing a line of code.

Graphical Screen Objects

Using the AG's WYSIWYG interface, you define input/output fields, menu bars, pushbuttons, check boxes, action buttons, scroll bars, scrolling lists, fancy text, multiple-line editors, and bitmapped regions. These objects will conform automatically to CUA rules through AG enforced standards. Productivity increases since the generation of every type of graphical object is done via the WYSIWYG tool.

Active Data Fields

With the Arcadia Model, a field is not a passive data container with user code that controls it. It is a fully functional control object that handles user interaction on its own without calling upon the application until there is user input to report. The Workbench takes care of retrieving screen variables and performing user-specified field entry validation actions. The Workbench variables will be bound with your application code variables at runtime.

Without writing a line of code, you can define and associate data validation rules with each field. Rules include ranges, data types, mandatory input, and translation rules. A message can be associated with each validation rule to guide the user through the data entry process. The Workbench allows you to bind variables (including arrays) to C, Pascal, or Cobol so that when a variable is updated in your code, it is displayed automatically in the panel, and vice versa. This allows input/output fields to have significant control over what information is transferred to the application. These features ensure both a very robust production environment for the end users as well as fewer hours of tedious programming.

Help

Using the AG environment, you create a hierarchy of help screens: from field-level, context-specific help to on-line documentation. You can assign a help window to a single visual object or to a collection of visual objects. At the broadest level, on-line documentation is provided through a Help Index that presents an alphabetized list of help panels that you have created. Help is even provided for the "Help" function. Any help panel can contain hypertext links into other help panels.

Help at all levels is in conformance with the CUA model. Users may obtain context-sensitive help at any time by pressing the F1 key. Help also is available through the menu bar.

For end users, this complete Help system provides ease of use and reduced dependency on what could be out-of-date documentation. Because help is always available at the point of use, users require less training and learn to use applications faster. For the programmer, there is no coding. In fact, the entire development of the help facility can be left to technical writers.

Messages

Using the AG, you can associate message pop-ups with any visual object. You specify the text of the message and assign it a type, such as notification, warning, or error. At runtime, the message and its appropriate CUA icon will appear when the conditions you specify occur.

Embedding messages throughout an application makes it more user friendly. Users can make decisions and correct mistakes on the fly. No programming is necessary, and this task, again, can be left to technical writers.

SCREEN LIBRARIES

Panels belong to libraries. At runtime, your application uses an API command to specify which library of panels will be used. This allows you to maintain different versions of the panels for different applications, different national languages, or to account for differences in the sizes of various displays.

Libraries allow panels to be reused by different applications, cutting down on development time. Several developers can work concurrently on the same project, using libraries to consolidate their work. Libraries reduce the size of the client code because the running application contains just the screens that will be used. For client-server applications, an added benefit is that the server can specify to the client which version of the library should be running. It also can download the appropriate version. This facilitates server-managed repositories of screen objects.

NATIONAL LANGUAGE SUPPORT

The Arcadia CUA Workbench helps you create built-in National Language Support for your applications, making them appropriate for worldwide use. All aspects of the user interface are translated and maintained easily. The five mechanisms that cooperate to provide this are:

- The unique architecture of the Workbench, which separates screen objects from the code, means that translations or modifications of the screen objects do not require you to recompile your client code.

The unique architecture of the Workbench separates screen objects from the code

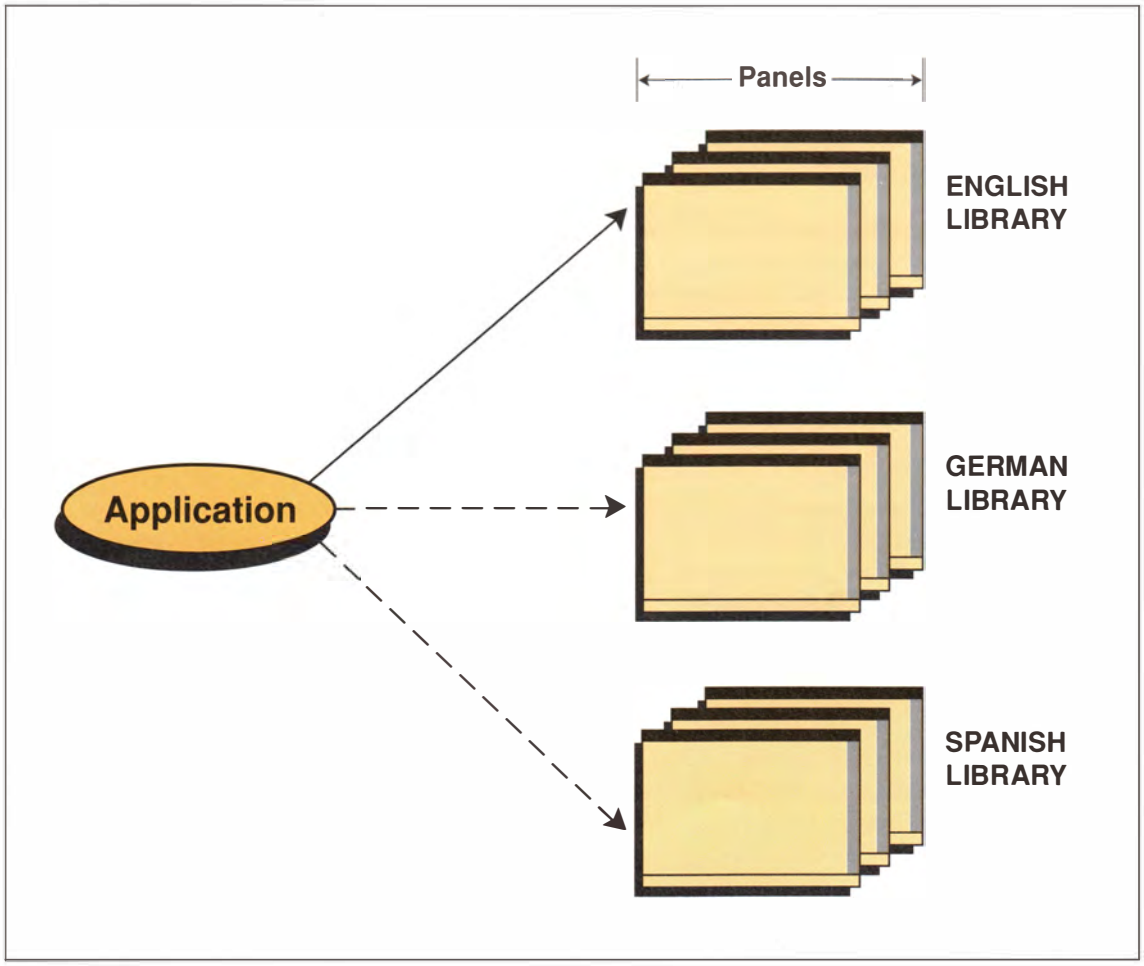


Figure 4. The Arcadia Runtime Environment

The AG has built-in facilities which make it easy to prototype even large applications quickly

- The screen library mechanism allows you to create separate libraries for various national language translations. At runtime, users, or even the server, can specify in which language the application should run (see Figure 4).
- The WYSIWYG tool (AG) has control over the final placement of all screen objects. You specify the region where you want objects to be placed, but the AG makes the final judgment to compensate for varying string lengths. This means, for example, that German versions of applications originally designed for English text will look correct without additional modifications.
- The AG generates an intermediate script language, based on the ISO SGML standard. A writer can use any standard ASCII editor to modify the script directly into a different language. Modified scripts can be viewed and altered, if necessary, using the WYSIWYG tool. Using the script instead of the AG for translation can be more productive since all translatable aspects of the GUI are located easily.
- The Workbench supports automatic time, date, and currency translation through the operating system's national language code pages.

MAINTAINABLE APPLICATIONS

The Arcadia CUA Workbench forces a natural separation of the application logic from the screen objects. This means that your application code does not contain the GUI code, and is therefore smaller, more structured, and easier to read and maintain. Visual objects may be modified without recompilation of the application code.

Because changes to the intermediate script language are reflected in the WYSIWYG tool, user interface maintenance can be done either from the AG or directly on the script using an ASCII text editor. Finally, the library mechanism allows different versions of applications to be maintained separately.

ARCHITECTED FOR PORTABILITY

The Arcadia CUA Workbench provides a scripting language that is independent from the GUI platform on which it runs. This unique architecture means that both the client code (which is written in a standard 3GL) and the graphical user interface are created only once, yet they can run on multiple platforms.

The creation of visual objects is a two step process. First, the AG converts the objects into the intermediate language. It is open and extensible, meaning that programmers can integrate additional GUI-specific features with minimal effort. Secondly, the Script Compiler compiles the SGML script into the appropriate runtime GUI code. Currently, applications written using the Arcadia CUA Workbench can be compiled into standard DOS, Windows 3.0, and Presentation Manager. Support for more GUIs, such as OpenLook, Motif, and Macintosh will be added.

RAPID PROTOTYPING

The AG has built-in facilities which make it easy to prototype even large applications quickly. After you use the AG to create panels, you can use the animate feature to compile and run the user interface in a separate window automatically.

The prototyping facilities allow users to navigate through the screens just as they would with the finished application. This provides an efficient means of obtaining a feel for how the system will work when it is completed. The prototyping facilities allow users to get involved at an earlier stage. Instead of a back-and-forth process of specifications and approvals, users actually can be involved in designing the look and feel of an application. No code needs to be developed in order to run the prototype, which saves the developer time and energy. These facilities also cut down on the lengthy specifications that usually are required when developing a large system.

CONCLUSION

Client-Server OLTP applications require new tools with unique features. The tools must combine flexibility with high performance, as well as powerful features with ease of use. They must run in highly complex graphical environments while maintaining portability across platforms and languages. The Arcadia CUA Workbench fulfills all of these requirements.

REFERENCES

1. Arcadia uses the same SGML script as OS/2's Dialog Manager™. Refer to GG24-3467 for an introduction to SGML.
2. *Systems Application Architecture Common Programming Interface Dialog Reference,* SC26-4356, defines the interface protocol on which the Arcadia CUA Workbench is based.
3. Orfali, R. and Harkey, D. *Client-Server Programming with OS/2 Extended Edition,* New York, NY: Van Nostrand Reinhold, 1991, pages 785-966. This book contains an excellent discussion of GUI tools for Client-Server applications. (IBM Order # G325-0650)

David C. Pacheco, *Arcadia Technologies, Inc., 735 West Duarte Road, Suite 207, Arcadia, CA 91007. Mr. Pacheco is the Technical Support Manager and has been with the company for two years. He received a BS in Computer Information Systems from DeVry Institute of Technology. He may be reached by phone at (818) 446-6945 or by FAX at (818) 447-4212.*

Arcadia Technologies Inc.

Pacheco can be reached by phone at (818) 446-6945 or by FAX at (818) 447-4212

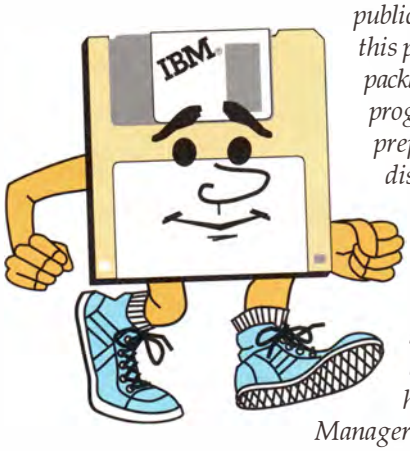
Software Tools

A Farewell To SneakerNet

Barbara Koob

by Barbara Koob

Do you want help in managing your workstation environment? Are you tired of the time it takes to install and update your workstation software? What about all those license agreements and keeping track of your inventory? If you have these problems, then bid farewell to SneakerNet because the IBM® SAA™ Delivery Manager is here! Announced last September, Delivery Manager is IBM's most recent program product for making your software administration, delivery, and installation of packages easier. (A package is any related set of files containing software, softcopy publications or just data.) Customers can use this product for distributing and managing packages from an SAA host to an OS/2® programmable workstation. Vendors can prepare their OS/2 products for electronic distribution using Delivery Manager. In June of this year, Delivery Manager became available for MVS/TSO and VM, supporting OS/2 workstations via LU2 communications. MVS/CICS™ support via LU6.2 communications will be available in December. So if you need help, think the IBM SAA Delivery Manager. It delivers!



CUSTOMER BENEFITS

Delivery Manager provides many powerful benefits to customers:

- **Ease of Use**
- **Relational Database Tracking**
- **Productivity**
- **Distribution and Scheduling**
- **Support for Both Push and Pull Environments**
- **Group Capabilities**
- **Authority Capabilities**
- **Version Control**
- **License Management**
- **Documented User Exits**

At the top of the list are easy to use administration functions. These functions are made accessible to both administrators and end users via the Common User Access™ (CUA™) interface. Delivery Manager administrators can register packages, enroll users, perform license management, and even schedule deliveries. Delivery Manager end users can install, update or delete packages on their workstations. Administrators and end users also can request online help for a task or unfamiliar term. If end users require help with these functions, there is an online tutorial to instruct them. Both administrators and end users need only an elementary knowledge of the OS/2 environment. System programming expertise simply is not necessary. Figure 1 is an example of the what an end user might see when installing a package.

Delivery Manager tracks software packages from initial purchase and approvals through distribution and maintenance, keeping an audit trail of all deliveries. The information in this audit trail is stored in a relational database, Structured Query Language/Data System™

(SQL/DS) for VM or Database 2™ (DB2) for MVS. Since the database views are published, customers may use these data to produce reports.

Delivery Manager totally automates the installation and updating of any OS/2 application. By eliminating repetitive distribution and installation tasks, customers will benefit in increased productivity. Delivery Manager not only provides for the delivery, but also completes the installation of a package. This is not the initial installation, but rather a cloning process which installs the product after it has been registered and customized for redistribution. (Distributing, installing, and updating OS/2 is a key customer requirement. Today, Delivery Manager can download OS/2 updates to a non-working directory just like any other package, but these files cannot be installed or applied at this time using Delivery Manager.) As part of the registration process, Delivery Manager allows the administrator to define the package profile, package status, target drive and directory, the package prerequisites, the startup information, and, of course, the package contents. When the registration process is completed, Delivery Manager uploads the new and changed files to the host for distribution to all authorized end users. Delivery Manager performs prerequisite checking prior to the download. It also updates the CONFIG.SYS file and the startup list.

Delivery Manager can operate in a centrally-controlled environment where an administrator can push packages to workstations. It also can work in an end user-controlled environment where users pull software from the host. This pull function is referred to as catalog shopping. (See Figure 2.) Either way, Delivery Manager provides for the authorization of users for specific packages. Delivery Manager also provides a scheduling function to allow packages to be distributed at a specific date and time. Scheduling of package deliveries can be done both by administrators and end users, even during off-shift hours. To help manage traffic, Delivery Manager has included host access times. Here an administrator may specify when an end user can access the host to pull packages down to the workstation. For ease of operation, Delivery Manager has a grouping capability. Administrators can create user groups and/or package groups for both authorization and scheduling. For example, an administrator may create a user group of all the personnel in the organization. If a fix to a package must be applied, all the administrator needs to do is update the package and schedule the updated package to the user group.

Another example is to create a package group of all the packages that a particular department must have. Every time a new user comes on board, the administrator simply authorizes and schedules that package group to the new user.

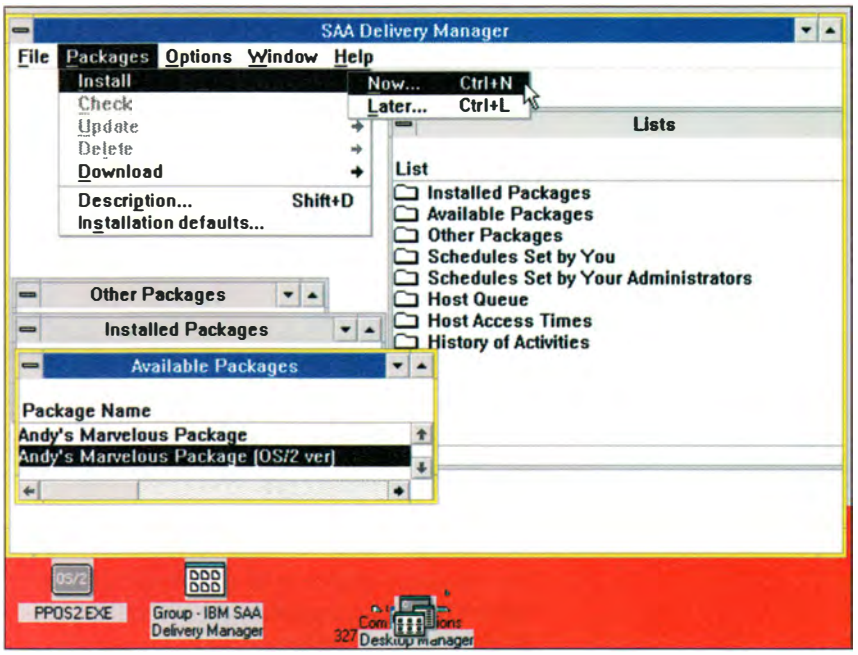


Figure 1. SAA Deliver Manager End User Screeen

Delivery Manager also allows customers the flexibility to define their own authority types. An authority type describes the functions that are authorized to a particular administrator. Some of these functions are: to create or update a user record, to create or update a software package, to grant access to a package, to schedule a package, or to manage license information. Customers may define multiple authority types with various functions. There may be one administrator who can handle everything or several administrators, each of whom has specific functions to do. One administrator may have the authority to control user records, while another administrator manages vendors and license agreements, and still another administrator builds software

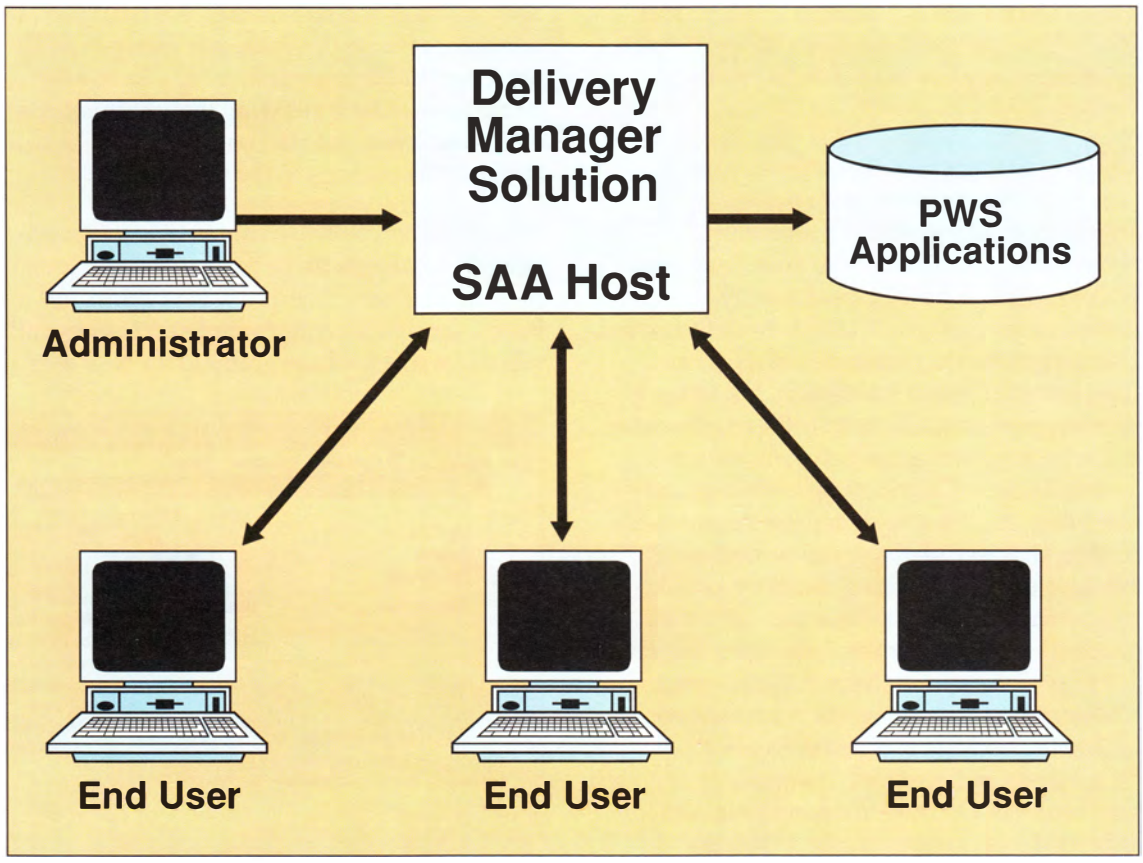


Figure 2. SAA Delivery Manager Provides Both "Push" and "Pull"

packages preparing them for distribution. It's the customer's choice as to how to set this up. This authority type processing even goes one step further. Administrators can have defined authorities but only over one or more specific objects. For example, an administrator can only grant access to users and packages that he or she *owns*. Or an administrator can only update license counts for packages for which he or she is responsible. This function is yet another way for customers to administer their software easily.

Version control is a prime function of Delivery Manager. Delivery Manager can store multiple versions, and even customized versions of a package on the host system, then authorize and deliver a specified version to a particular set of users. It handles both updates and upgrades of packages where only the changed files are downloaded. In this scenario, "update" refers to a package modification or fix, and "upgrade" refers to upgrading to the next package version. Administrators can view online which version and modification level is installed for each package and end user, or produce reports from the database to gather that information. As updates are made to a specific version of a package, Delivery Manager keeps track of the changes. Because you can allow users to have full control of their workstation, users may have various levels of a package update. In this case, Delivery Manager will replace only those files that have changed. (See Figure 3.)

Managing vendor license agreements is yet another significant function of Delivery Manager. To accomplish this task, Delivery Manager stores key information about the vendor's product and licensing agreement. As installations and upgrades take place, an entry is logged in the audit trail that relates each delivery with a particular vendor, product, contract, and license. Delivery Manager can count the number of copies that have been installed and stop distribution when a maximum count is reached. Alternatively, if a

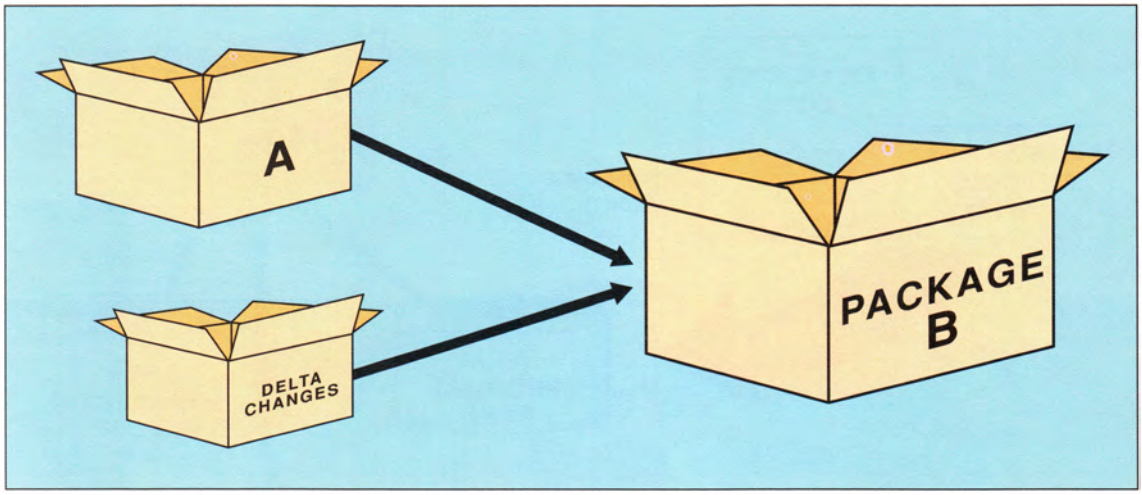


Figure 3. Package Versioning

vendor licensing agreement allows, the customer may desire to continue distribution of a package regardless of any quantity negotiated. Here, vendors are paid after the fact. This process is known as post-ordering and can be used by customers to *pay as you go*.

If you wish to tailor your delivery management system, a series of documented user exits are provided with Delivery Manager. Customers can write their own programs enhancing Delivery Manager functions to meet their individual business needs. Some examples of possible enhancements are: using the threshold count user exit to notify a purchase administrator that the maximum number of copies is almost reached, feeding software asset data into the customer's own billing system, synchronizing a schedule of package deliveries to prevent them from being sent all at once, rebooting the operating system in the event of configuration changes, or calling a virus scanning program. For interface details of these user exits, refer to the document *IBM SAA Delivery Manager Programmer's Reference*, document number SC34-4324.

Delivery Manager is a cooperative processing application which runs on both the host and the workstation, using the Application Connection Services (AConnS) product for distribution. OS/2 EE 1.2 is required for users of Delivery Manager. Administrators must have 1.5Mb of RAM and 4.0Mb of DASD. End users must have 1.0Mb of RAM and 3.5Mb of DASD. A total of 10Mb of RAM is recommended to support OS/2, AConnS and Delivery Manager. 10Mb of DASD is recommended to support AConnS and Delivery Manager.

Certainly developers can save valuable time, resources and money by managing, installing, and maintaining their application software assets electronically, not to mention by using the capabilities to redeploy assets, set company standards, and be better able to aggregate licenses for volume purchase savings! For more information about the IBM SAA Delivery Manager and customer benefits, you can order the publication entitled *SAA Delivery Manager Introduction,* document number GC34-4190-0.

VENDOR BENEFITS

Notwithstanding all of the customer perks, software vendors can benefit as well! Some of the top benefits include:

- Savings on distribution and support costs
- Increased revenue via catalog shopping
- Electronic installation flexibility
- Help with tracing unauthorized copies

Since vendors no longer would need to create and distribute shrink wrap for new versions, upgrades or updates, they can save tremendously on their distribution and support costs. In addition, vendors can increase their

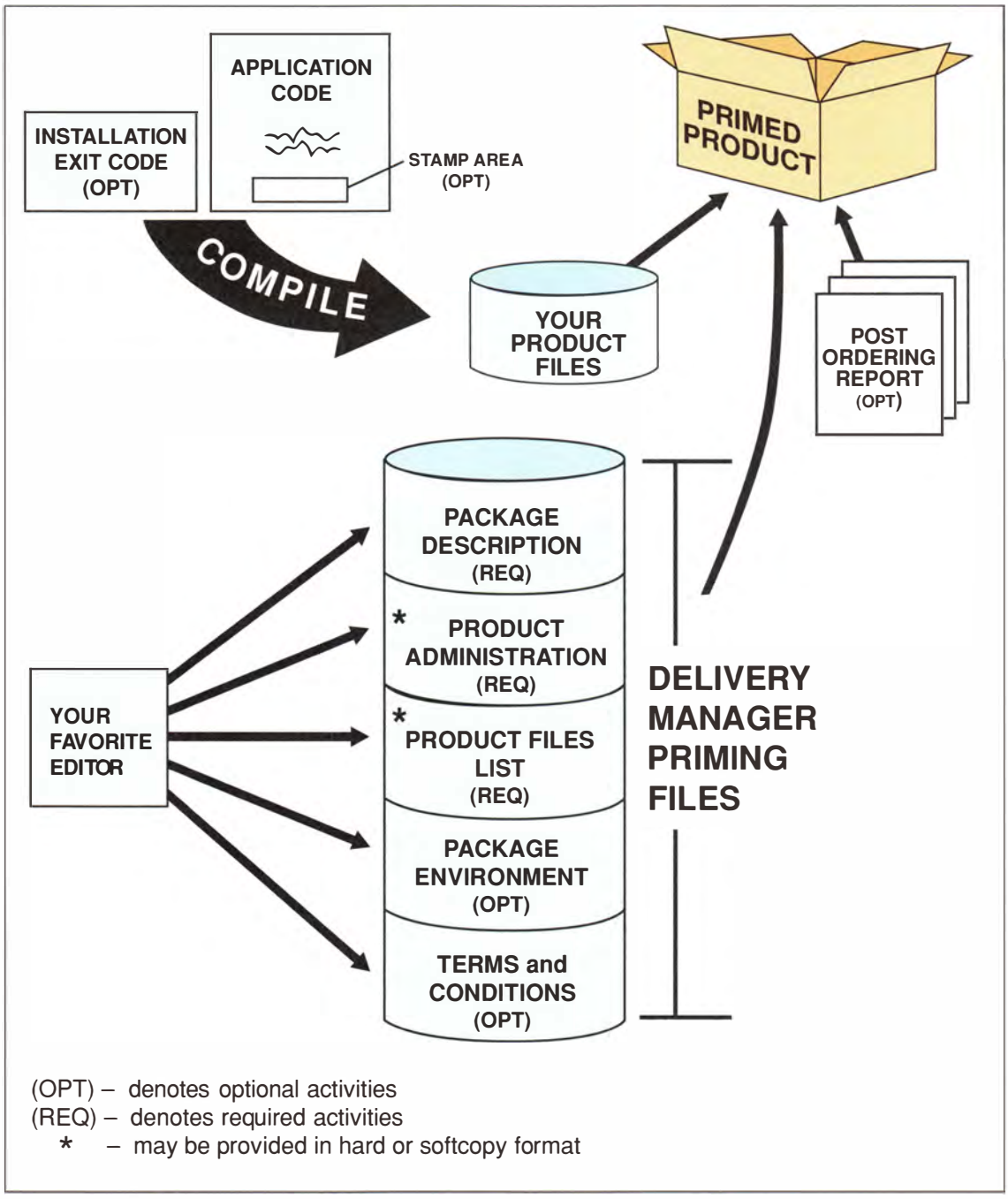


Figure 4. Preparing Your Product for SAA Delivery Manager.

revenue via the Delivery Manager catalog shopping feature. When coupled with the post-ordering process, large customer enterprises potentially could generate even more sales. However, even if a vendor does not implement a post-ordering process, Delivery Manager still can be used to make your products more visible to potential users. Delivery Manager also allows vendors to tailor their installation processes for electronic delivery. Via the installation user exit, vendors can write programs to tailor their unique installation situations. Such a user exit program, for example, could perform product-specific installation options for a target workstation, bind a database, execute a product registration process, or even implement a rental agreement for that software. Another vendor benefit is the ability to trace unauthorized distribution of your software product. To do this, Delivery Manager allows the creation of a stamp area in one or more of the files associated with the

product. On delivery, Delivery Manager places identifying information into this stamp area. This information is masked so that no one can see it or change it easily. While this function will not prevent unauthorized copies, it does help in determining the original source.

PREPARING YOUR PRODUCT FOR DELIVERY MANAGER

Preparing your product for Delivery Manager is comparatively simple. It is not necessary to have Delivery Manager as a prerequisite, nor does using Delivery Manager preclude the use of another delivery vehicle. However, it will save your customer setup time and make your product easier to distribute and install. But how do you prepare your product? Figure 4 depicts this process.

Only three activities are required to prepare your product for the SAA Delivery Manager:

1. Creating a package description file
2. Creating product administration data
3. Creating product files information.

- The package description file contains a short informative description of your product. Delivery Manager end users can read about this product by viewing the contents of this file. By creating package description files, vendors create new opportunities to help market their products to Delivery Manager users.
- Product administration data refers to the ordering information about the product. This information is used both by customers and by Delivery Manager to monitor software inventory. You can provide this information in either hard-copy or soft-copy format.
- Product files information refers to the files contained in the product, and the recommended subdirectory structure for installing the product. This information is used to help the Delivery Manager administrator register a package and its contents. It also may be provided in hard-copy or soft-copy format.

There are six other optional activities which you can use to prepare your product for electronic distribution:

1. Creating a package environment file
2. Creating a terms and conditions file
3. Providing a stamp area
4. Writing an installation user exit program
5. Creating a post-ordering report
6. Testing the distribution of your product.

- The package environment file is used by Delivery Manager to rebuild each user's CONFIG.SYS file automatically when a product is installed, updated, or deleted. If your product requires command statements in the CONFIG.SYS file, you should create a package environment file.
- The terms and conditions file contains the license agreement for the product. The contents of this file may be displayed to users who install the product. It must be used if your license agreement requires users to accept the terms and conditions. In most cases, however, it is the customer, and not the end user, that accepts a vendor's terms and conditions.
- If you want the Delivery Manager stamper program to place identifying information which is masked from the end user, you must include a stamp area in one or more of your product files.
- As described earlier, the installation user exit may be used to tailor part of your product's installation process.
- If you have implemented a post-ordering process with your customers, you can create a post-ordering report using the published Delivery Manager data views. Customers can use your report to produce output for orders into your business systems, thereby generating additional revenue.

- Finally, if you have installed Delivery Manager, you should conduct a trial installation of your product. This test is especially important if you have implemented the asset stamper or installation user exit activities.

For more information on preparing your product, you can order the publication Preparing Your Product for the SAA Delivery Manager, document number, SC34-4198-0. Another book that you may find helpful is the IBM SAA Delivery Manager Administrator's Guide, document number SC34-4191.

CONCLUSION

So if you're tired of trying to manage your workstation environment, or you want to prepare your software products for electronic distribution, bid farewell to SneakerNet today! SAA Delivery Manager delivers!

REFERENCES

IBM SAA Delivery Manager Administrator's Guide, SC34-4191.

IBM SAA Delivery Manager Introduction, GC34-4190.

IBM SAA Delivery Manager Programmer's Reference, SC34-4324.

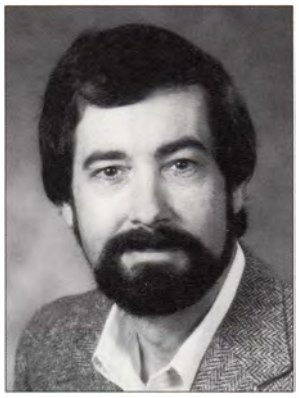
Preparing Your Product for the SAA Delivery Manager, SC34-4198.

Barbara Koob *is an advisory product planner for the SAA Delivery Manager product. She graduated from Western Connecticut State University in 1978 with a BS degree in Mathematics, and joined IBM in 1979 in Poughkeepsie, N.Y. as an applications programmer, working in development of IBM Information Systems Management products and internal support tools. She was the technical leader for the IBM IPS Support System and Electronic Software System projects. In 1984, she transferred to Cary, N.C. into product planning working on customer requirements, project management, translation planning, and technical planning. In 1988, she joined the SAA Delivery Manager planning team. In the last three years, she has presented IBM's approach to software distribution to numerous customers and vendors. She also designed, marketed and implemented a vendor program for enabling vendor software for electronic distribution using the SAA Delivery Manager.*

Software Tools

Debugging OS/2 1.x Device Drivers Using Periscope/ Remote for OS/2

by Brett Salter



Brett Salter

INTRODUCTION

Software developers writing applications for OS/2® have had full-screen, source-level debuggers available to help them debug their applications for some time. On the other hand, developers writing device drivers for OS/2 have had to work with much more primitive tools until the recent release of a full-screen, source-level debugger for OS/2 device driver development: Periscope®/Remote for OS/2. Developed by The Periscope Company, Inc. of Atlanta, GA, an established vendor of debuggers for PC-DOS, this new debugger supports the development of both Base and PM drivers under OS/2 1.x. It provides software-only or real-time hardware-assisted debugging capabilities, depending on the model of Periscope with which it is used.

ADVANTAGES

This new OS/2 debugger has some significant advantages over other device driver debuggers, including:

- Enhances productivity via full-screen, symbolic and source-level debugging.
- Provides easy access to any memory location in the system.
- Has minimal impact on the system being debugged.
- Coexists with a Ring 3 applications debugger.
- Offers software-only and real-time hardware assisted debugging.

DESIGN STRATEGY

To debug device drivers, the Periscope/Remote software must be available at CONFIG.SYS time, meaning it must be a device driver. By running as a driver, it has the advantages of running at Ring 0, and the resulting ability to access any memory or I/O port in the system.

Since device drivers use memory in the first megabyte of system memory, any space they use subtracts from the space available for the DOS compatibility box. So, to minimize the size of the debugger in the OS/2 system, Periscope/Remote is configured as a small (4K) device driver. It runs on the OS/2 system, where the device driver being debugged runs. This target OS/2 system is connected by a null-modem cable to a host PC-DOS system where the *regular* Periscope software runs.

This approach also keeps the impact of the debugger on the target system to a minimum. Since the Periscope/Remote device driver's purpose is limited to reading and writing memory, registers, and I/O ports at the behest of Periscope in the host system, it uses a minimum set of resources in the OS/2 system. This set includes the usual debugger interrupts (1 - single step, 2 - NMI, and 3 - code breakpoint); a single communications port (which may be configured as COM1 through COM8); and optional exception conditions, such as the infamous Trap D. Although it requires two systems, the remote approach has

a significant advantage. By minimizing the size and side-effects of the debugger, bugs are less likely to disappear or change characteristics when the debugger is used.

Even though Periscope/Remote is intended for driver-level debugging, it also can be used at Ring 3 for applications-level debugging. In most cases, however, applications will be debugged more easily using a product such as MultiScope™ or CodeView™. Since the Periscope driver ordinarily presumes that it 'owns' the debugger interrupts, there is a command in Periscope to allow it to share the debugger interrupts with a Ring 3 debugger. In this way, you can use the application debugger to debug the application program that communicates with the device driver while the driver itself is being debugged by Periscope. Using this technique, you can maintain control over both ends of the debugging session.

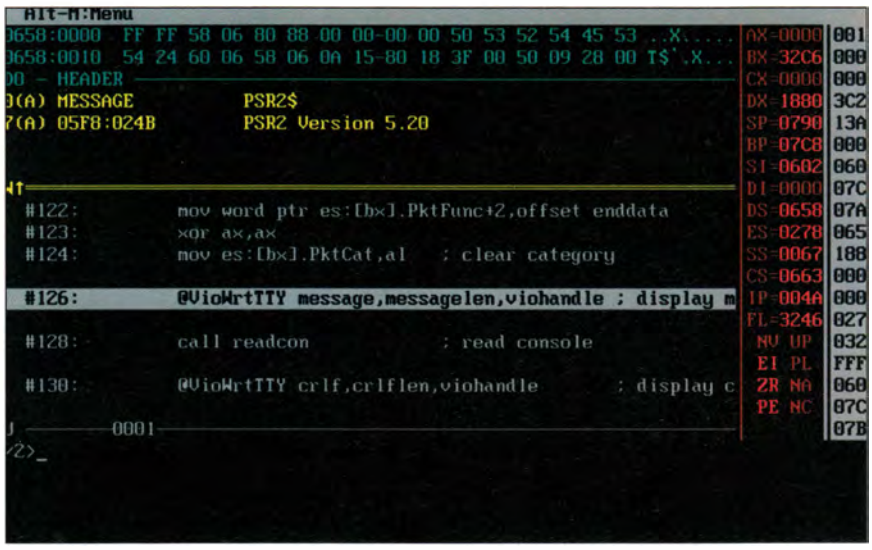


Figure 1. Periscope's screen showing source code for an OS/2 device driver.

Periscope Model IV is a hardware-assisted debugger that monitors the CPU of the target system, and can be used as the host Periscope software. In addition to providing software debugging capabilities, this product allows you to set real-time breakpoints on memory or I/O accesses and to capture a real-time execution history of the system in a circular trace buffer. It is available for systems whose CPU is an 80286, 80386DX, 80386SX, or 80486DX at speeds up to 33 MHz. The real-time trace buffer is especially valuable since it lets you see what led up to a breakpoint while your program runs at full speed. For some of the mysteries of protect mode programming, capturing a trace history and analyzing the result may be the only way to understand what has happened in your system.

Both the software-only products and the hardware-assisted products can use the non-maskable interrupt (NMI) to activate the debugger. This signal can be generated either on the host or on the target system to wake up the debugger via the Periscope break-out switch. Or, for those of you who may have an old board from the IBM® Resident Debug Tool (RDT) lying around, you can dust it off and put it back into service!

GENERAL DESCRIPTION

Periscope/Remote for OS/2 is a separate software package that is used with any of the standard Periscope models (software version 5.2 or later) to create a full-function source-level debugger for OS/2. It supports source-level debugging of various languages, including Assembler, BASIC, C, C++, Fortran, Pascal, and others from various vendors. In addition to the source-level support, it allows access to global and local variables.

Periscope's screen (see Figure 1) defaults to showing a menu bar across the top line, followed by horizontal data, watch, and disassembly windows. The data window can display memory in any of 15 formats, including byte, word, double word, ASCII, and others. The watch window can display multiple different memory locations in any of the display formats. The disassembly window disassembles memory in one of three formats: source only, assembly only, or a mixed mode.

On the right-hand side of the screen are the vertical register and stack windows. Changed registers are highlighted so you easily can see which registers were modified since the last time Periscope's screen was displayed. The stack window shows the bottom of the stack (SS:SP) at the top of the window, with the succeeding lines representing higher addresses.

You can change the screen easily. The presence, order, size, and color of each window are user-selectable. For example, when examining memory, you may wish to change the screen temporarily to display one large data window in ASCII format.

Periscope's native mode is as a command-driven debugger with commands similar to those of DEBUG and SYMDEB. It can be used with pull-down menus — the best method for the new or casual user to access the many commands available with the debugger.

The command summary (see Figure 2) shows Periscope's extensive set of functions, which includes memory, register, and I/O port manipulation. There are a variety of breakpoints, including code, debug register, and conditional or evaluated breakpoints. These allow you to describe breakpoint conditions to the debugger. When using the hardware-assisted models, additional real-time breakpoints are available. Periscope has powerful search commands that support the usual string searching, and also allow you to search for memory references, disassembly strings, and to search the stack to display the return addresses of calling procedures. Of particular interest are the ability to use the debug registers, the ability to display memory anywhere in the system, and the descriptor displays, including the Global and Local Descriptor Tables (GDT and LDT) and the Interrupt Descriptor Table (IDT).

The debug registers are features of the 80386 and 80486 CPUs. These registers allow you to set up to four breakpoints on code execution or memory accesses of up to four bytes in length. For example, if you need to set a code breakpoint in ROM or stop when a particular word of memory is written, the debug registers will allow you to execute your program at full speed while waiting for the event to occur.

Periscope/Remote is a debugger for device driver developers

?	-	Help.
A	-	Assemble instructions into memory.
Bx	-	Set breakpoints, including code, debug register, and a variety of conditional breakpoints.
C	-	Compare two regions of memory.
Dx	-	Display memory in fifteen different formats, including GDT/LDT and IDT displays.
Ex	-	Modify memory, define symbols.
F	-	Fill memory with a pattern.
Gx	-	Various Go commands that execute until a breakpoint is reached.
Hx	-	Hex arithmetic and real-time hardware specific commands.
Ix	-	Read a value from an I/O port.
Jx	-	Step over the next instruction or until the next source line.
Kx	-	Repaint the debugger screen.
Lx	-	Load various files, including batch, definition, and symbol files.
M	-	Copy memory from one location to another.
Ox	-	Output a value to an I/O port.
Qx	-	Quit the debugger. Includes options to reboot the host and target systems.
Rx	-	Display, modify, save and compare system registers.
Sx	-	Search memory in a variety of formats.
Tx	-	Trace execution and display the software trace history.
Ux	-	Disassemble memory in three formats, including source only, assembly only, or a mixed source and assembly format.
Vx	-	View a source file.
Wx	-	Write various files, including batch, definition, and symbol files.
Xx	-	Translate hex numbers to decimal and vice-versa.
/x	-	Options to switch symbol tables, echo Periscope's output to a disk file, capture keystrokes to a disk file, display nearest symbols, perform symbol manipulation, perform user exits, customize Periscope's screen display, etcetera.

Figure 2. Summary of Periscope/Remote Commands

Periscope/Remote for OS/2 allows you to access memory anywhere in the system. This can be useful when you need to display or modify memory, but don't have a selector that points to the desired address. For example, if you need to display memory addressed by the BIOS data area in low memory or to disassemble memory in the ROM BIOS at the end of the first megabyte, you can use a command to set up a GDT selector previously reserved by Periscope that can be used to access the desired memory region.

In protect mode, your driver must use selectors instead of segments for all memory accesses. The ability to display both the Global and Local Descriptor Tables (GDT and LDT) provides valuable information when debugging a protect mode program. For each selector, you can see the base, limit, access rights, present and accessed bits, default privilege level, and a selector description. Similarly, the Interrupt Descriptor Table (IDT) display shows you the gate type, the selector and offset, access rights, present bit, and the default privilege level.

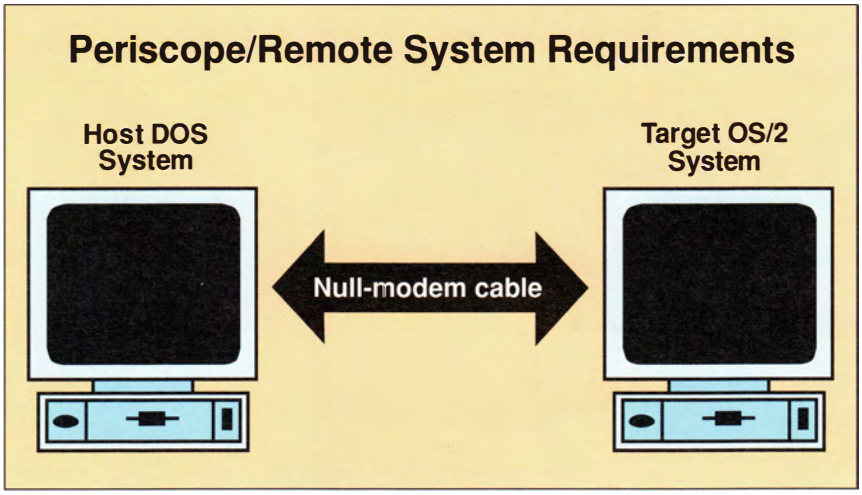


Figure 3. Periscope/Remote System Requirements

Host System
- *Must be running PC-DOS or MS-DOS.*
- *Periscope software runs on this system.*
- *Source code and symbol file for your driver must be on this system*

Null-modem Cable
- *Transmit & Receive lines must be crossed.*
- *CTS & RTS lines must be crossed.*
- *Ground must carry straight through.*

Target System
- *Must be running OS/2 1.x.*
- *Periscope driver runs on this system.*
- *Executable code for your driver must be on this system.*

SYSTEM REQUIREMENTS

The system requirements for running Periscope/Remote for OS/2 are illustrated and described in Figure 3.

The host system must be a PC-compatible machine running PC-DOS version 3.00 or higher. Due to subtle incompatibilities of the DOS compatibility box under OS/2 1.x, you cannot use the DOS box — it must be a real copy of DOS. The system should have a hard disk with approximately one megabyte of available disk space. The display can be a monochrome or color display of any type. Since this system will run in real mode, no extended memory is needed, but the system should have a full 640K of DOS memory. One COM port is needed for the remote communications to the target system. The host system may be anything from the original IBM PC to the latest PS/2® Model 95.

The main Periscope software runs on the host system. The source code and symbol file for the driver being debugged must be present on the host system for Periscope to support source-level debugging.

The host system is connected to the target system via a 5-wire null-modem cable. You either can make your own cable, or use the one that is supplied with the software packages Brooklyn Bridge™ or LapLink® III.

The target system must be running OS/2 1.x. As noted above, one COM port is needed for the remote communications to the host system. Since it is running OS/2, obviously the host system must have an 80286 or later CPU. The Periscope/Remote for OS/2 driver, PSR2.SYS, is installed on this system. The executable file of the device driver you are debugging also must be present on the target system.

INSTALLATION

Installing Periscope/Remote is simple and straightforward. It involves copying the requisite files to the target system, and then making a few changes to the CONFIG.SYS file. Installation options include: COM port select, baud rate, enable/disable debug register, interrupt handling, and others. Installation concludes with the loading of the main

Periscope program on the host system, selection of *remote* mode at the host, and rebooting the target system so that the changes to the CONFIG.SYS file will execute.

USING PERISCOPE/REMOTE

The installation package includes a sample program called PAUSE.SYS, which illustrates the use of a remote debugger. It is invoked automatically at the end of the installation process. Figure 4 contains a portion of the source code.

PAUSE.SYS is a simple program that pauses the system and prompts the user to press a key to continue execution. To stop in Periscope, we've embedded an 'INT 2' instruction in the initialization phase of this driver (see Figure 4). Either an INT 2 (software NMI) or an INT 3 (code breakpoint) can be used to wake up Periscope easily. You also can use an illegal instruction, double fault, or general protection fault to activate the debugger if you prefer.

Once you've stopped in the driver, you'll be sitting at the instruction after the INT 2, but you won't have any symbols or source-level support, since Periscope doesn't know what

```
init    proc near               ; last thing in code segment!
        int 2                   ; stop in Periscope

        mov ax,word ptr es:[bx].PktFunc
        mov word ptr DevHlp,ax
        mov ax,word ptr es:[bx].Pktfunc+2
        mov word ptr DevHlp+2,ax

        mov word ptr es:[bx].PktFunc,offset init
        mov word ptr es:[bx].PktFunc+2,offset enddata
        xor ax,ax
        mov es:[bx].PktCat,al   ; clear category

        @VioWrtTTY message,messagelen,viohandle ; display message

        call readcon            ; read console

        @VioWrtTTY crlf,crlflen,viohandle       ; display crlf

comment|
        ; the following code will cause a trap d!
        push bx
        push es
        xor bx,bx
        mov es,bx
        mov bx,es:[bx]          ; stops here!
        pop es
        pop bx
end comment   |

        mov ax,0100h
        ret
init    endp
```

Figure 4. A portion of the source code for the sample device driver PAUSE.SYS

symbol file to use and where to locate it (see Figure 5). Using two Periscope commands, you can load the symbol file on the host system and then relocate it based on the current values of the code segment (CS) and the data segment (DS). Once this has been done for a particular driver, the command sequence can be set up as a keystroke macro, since it would change only as a result of major rework to the structure of the driver.

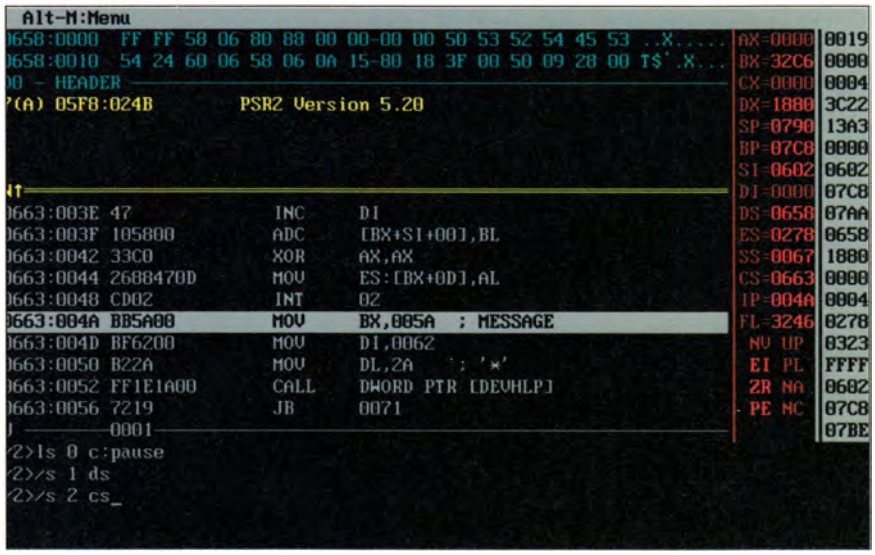


Figure 5. Periscope's screen showing the assembly code for an OS/2 device driver. Compare this screen with Figure 1.

At this point, you'll have full symbol and source-level support (see Figure 1). You can now step through your driver at the source level and display program variables using symbol names instead of absolute memory addresses. Even for hard-core assembly language programmers, the availability of source code with comments is a tremendous productivity enhancement, since you don't have to guess what the code is doing. You can see your actual source code while stepping through it, which helps you get and keep your bearings. The same is true for program variables, since you don't have to calculate their locations. You can just use the symbol names to access the desired data.

The debugging of PM drivers is similar to that of Base drivers, except that PM drivers do not run at Ring 0 or use GDT selectors. They run at Ring 2 and use LDT selectors, which means that you won't be able to access the code or data when the parent LDT is not active. To activate Periscope from a PM driver, you'd use an embedded INT 2 instruction (see Figure 6).

```
#ifdefined(PERISCOPE)
    _asm{int2}//call Periscope
#endif
```

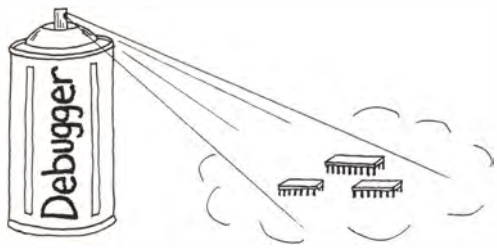
Figure 6. The "C" source code needed to activate the debugger via an INT 2

CONCLUSION

Periscope/Remote for OS/2 provides a full-function source-level and symbolic debugger for device driver developers. This product can improve programmer productivity dramatically since it allows the assembly language or C programmer to debug a device driver at the source level, and to reference program variables by name. For speed-sensitive or real-time applications that need ICE-like capabilities, the hardware assistance provided by the Periscope Model IV can capture real-time execution history. This postmortem analysis shows exactly what occurred in the system and eliminates time-consuming guesswork.

The two-system approach to debugging is very extensible and readily movable to new environments. With current support for DOS, OS/2 1.x, and Windows® 3.0, support for OS/2 2.0 is in development and should be available by the time this article is published.

Brett Salter *is the founder and President of the Periscope Company, Inc. He graduated from Georgia Tech in 1973 with a Bachelor of Mechanical Engineering degree, and has worked with PCs since 1981, writing assembler subroutines even before MASM 1.00 was released. He serves as president and chief developer, and directly oversees the company's software and hardware development activities.*



Presentation Manager

Printing Using OS/2

by David E. Reich

The OS/2® Print Subsystem has received much press over the last year or so. It is a very powerful mechanism providing applications freedom from needing to know what hard copy devices are present, what their capabilities are and whether or not they are busy with jobs.

OVERVIEW

The core of the OS/2 Print Subsystem is the spooler. A spooler is a program that intercepts data destined for a device (in this case a printer) before the data actually reaches the device. This is known as *spooling* the data. Multitasking is a primary reason for having a spooler. If more than one application is trying to print to the same device at the same time, each program's output would be intermixed with the other. With the spooler in place and active, programs send data to what they think is a printer, but is in reality the spooler. Once the print job is completely sent from the application (completely spooled), the spooler mechanism takes over sending the data to the physical printer. Thus, the spooler manages many applications sending print jobs at the same time to the same device.

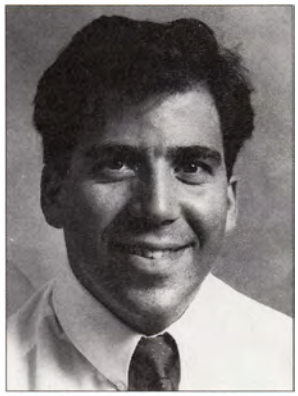
The spooler is only one part of the OS/2 Print Subsystem. It is a network of components, each performing a specific function. Figure 1 illustrates an overview of the Print Subsystem.

The main components are the Print Manager, spooler and printer drivers.

The primary user interface to the Print Subsystem is a program known as the Print Manager™. This is a Presentation Manager™ program that controls the user-modifiable aspects of OS/2 printing. These include installation of printer drivers, creation and deletion of queues and setting features such as the printer timeout values and paper orientation.

The spooler is the workhorse. It controls the order in which jobs are printed and how the operating system interacts with the physical hardcopy devices.

The final part of the Print Subsystem is the printer driver. A printer driver serves several purposes. It is the mechanism by which OS/2 accomplishes device independence with respect to printing. Device independence simply means that the application programs need not know anything about the device upon which their data is to be displayed.



David E. Reich

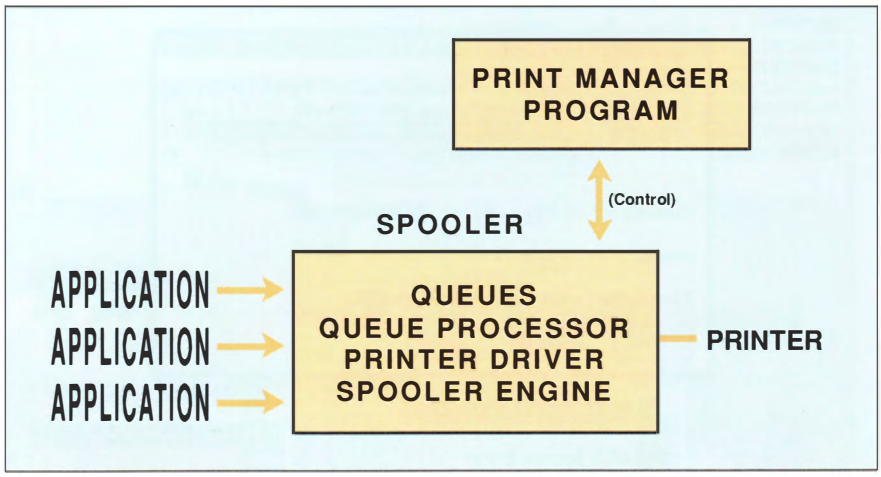


Figure 1. Overview of the OS/2 Print Subsystem

The OS/2 Print Subsystem gives applications freedom from knowing what printers are present

Applications send output to a generic presentation space. This presentation space is associated with a device context. The device context is not a physical entity, but more a logical description of the physical device.

In the world of printing, the device context is the printer driver. The printer driver does the translation from the device-independent form sent to it by the application and the Presentation Manager into a data stream understandable by the printer. At this level, the printer driver will interpret the drawing commands and turn them into commands the printer can understand.

SETTING THINGS UP

There are many print options that can be set by the OS/2 user. The discussion will start from the ground up.

To begin with, a printer must be connected to a physical port on the computer. This can be a parallel (LPTx) or a serial communications (COMx) port. Often, the type of communications is dictated by the printer. For example, an IBM Graphics Printer cannot be attached to a serial port. In other cases, the user can decide which method of communication the printer uses.

Once the port is decided on, the user must determine the correct printer driver to use. IBM supplies a variety of printer drivers with the OS/2 package. Included are drivers for the Hewlett-Packard Laserjet® series, the IBM Proprinter® series, plotters and Postscript® printers. In many cases, a single printer driver supports several flavors of each printer. For example, the Plotter driver supports the IBM 6182™, 7475™, 6180™ plotters and the Hewlett-Packard 7550A™ among its list.

After the exact type of printer and the model is determined, a printer driver for it may have to be installed — unless it was done when the system was installed.

A new feature of OS/2 1.3 is the Printer Installer. (See Figure 2.) The user can invoke this program by pulling down the Setup menu and selecting "Printer Install..." This will bring up a dialog with some simple questions. The program does the rest. The following is step-by-step walk-through of the process:

- Check the list of drivers already installed to see if the required driver is already installed. If not, insert the first printer driver diskette included with OS/2 into drive A.
- Select the "New..." pushbutton on the dialog. The default is the "A:" diskette drive. Select the OK button and the printer installer will read the diskette looking for drivers. When it finds them all, it will add them to the list. If the required driver is still not there, repeat the procedure until the disk with the desired driver is inserted.
- A shortcut to this method is to copy all of the files on the printer driver diskettes to a scratch subdirectory on your hard disk. Then when asked where your printer drivers are, simply overtype the drive and subdirectory (which lists "A:" as a default) with the correct directory. Reading the drivers from the hard disk is much faster than from a floppy.

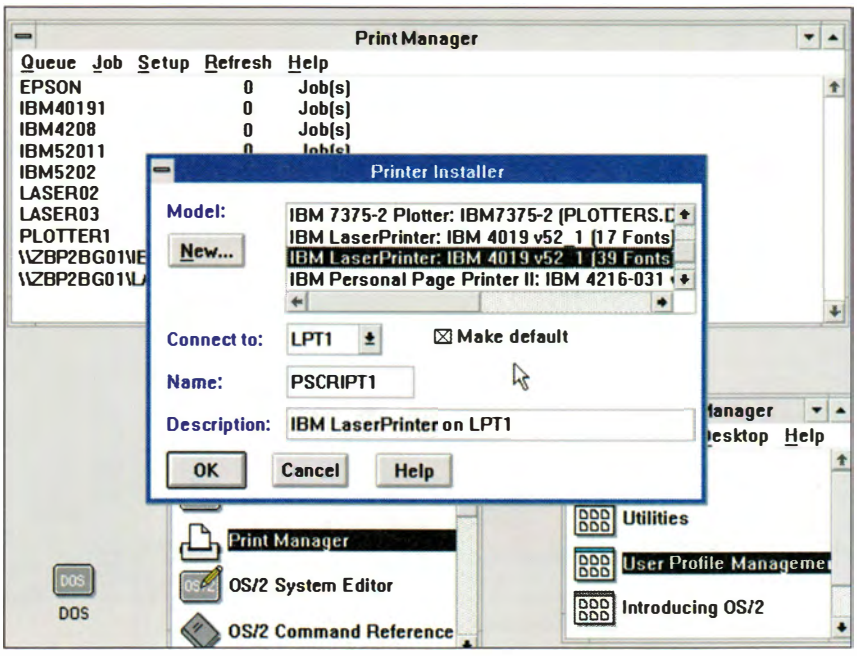


Figure 2. The Printer Installer

- Enter the items in the dialog according to desired printer set up. Specify the printer model, the port it's connected to, a name for the printer and a textual description. Once this is done, select OK and the program will associate that driver with the specified port, and connect a queue to that driver.

QUEUES

A queue is a virtual waiting line, much the same as a line at a bank. As jobs are sent to the spooler, destined for a particular queue, they will line up. As each job is processed, the others move up in the queue until all jobs are finished. One may ask why is there a queue in the middle? Why not just line up all jobs on the printer and let the spooler send each one in turn?

The most prominent reason is that as a feature of OS/2 printing, several queues can be associated for each printer driver. Although only one printer driver may be attached to a port, it may have many queues associated with it. Each queue can be configured with a unique set of properties, such as scaling and paper orientation. In this way, the user doesn't have to change the printer properties between jobs. There can be two queues, both hooked up to a single printer driver, each with unique characteristics.

PRINTING JOBS

Printing may differ slightly from application to application, but for the most part, it works the same way in all Presentation Manager programs. Usually there is a dialog specific to printing. Most times, it can be found on the "File" pulldown of the application. Selecting the "Print" menu item will bring up a dialog to specify print options. The choices include printer names and/or queue names, number of copies and paper orientation. This dialog will differ from program to program as each may choose to utilize certain functions but not others. For example, a word processing program may use headers and footers, but a drawing program for artistic or design purposes may not have a use for them.

Once the choices have been made, an "OK" pushbutton initiates the print job. The job begins spooling to the specified device/queue. At this point, the application is finished printing and may continue on to other work. This is another feature of OS/2 multitasking. The application will print the job, and then, rather than having to wait for the printer to finish, the application lets the spooler handle the actual hardcopy while it can go on to other work. Once the job is finished spooling, the second phase begins: the spooler takes the job and sends it to the device.

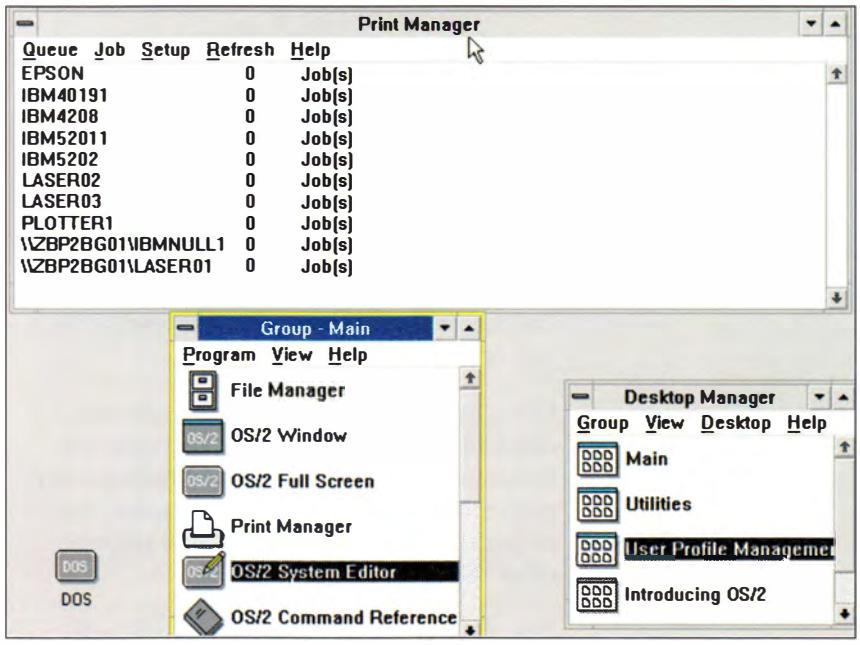


Figure 3. A Typical Print Manager Window

USING THE PRINT MANAGER

A user can look at, change and cancel jobs even after they have been completely spooled. This manipulation is accomplished with the OS/2 Print Manager program, the user's visual interface to the spooler. Figure 3 shows what a typical Print Manager window looks like.

The client area of the Print Manager contains all of the printers and queues that are currently available for use. This includes local printers as well as network printers. Contained in this area is the queue or printer name, the number of jobs currently in that queue and the queue status. Included in the possible values for status is "Queue Held" and "Printer Jammed" so that the user can see what state the printer is

in, if it needs service and so on. If there is nothing in the status area, the queue is in a normal, functioning state.

This area functions like a listbox, using the mouse or keyboard. The first menu is for queue manipulation. The available options are shown in a normal color while the unavailable ones are greyed. The options are to hold a queue, release a queue and to cancel all jobs on a queue. If an item is greyed, it is because that operation would not have any effect on the queue. For example, there is no point in cancelling all jobs on a queue that has no jobs in it, or releasing a queue that is not held.

The next item is the "Job" menu. This allows the user to manipulate individual jobs. Jobs on a queue are seen right underneath the queue name in the Print Manager window. Users can select a job in the same way they select a queue. Any operation they choose with the "Job" pulldown will be executed on the job(s) highlighted. Again, unavailable operations will appear greyed.

Using this pulldown, the user can display details about a job. For example, when this option is selected, they will be shown the size, printer driver, date the job was created, the physical device and current status among other things.

The "Cancel" option is exactly that; it will cancel the selected job. There is one instance where a cancel may not be able to be effected: plotter jobs. Because of the nature of the plotter driver, it is often impossible to cancel plot jobs once they have started printing.

The next two options allow the user to reorder jobs. "Print Next" and "Start Again" do exactly what they appear. When the user selects a job to be printed next, it will begin as soon as the current job, if any, is completed. The last two options are analogous to holding and releasing a queue, but in this case they are for individual jobs.

Next is the "Setup" pulldown. This is where the user can configure the various aspects of the print subsystem. The first item is "Printer Install" which has already been discussed.

The second item in the "Setup" pulldown is the spooler enable selector. When this item is selected, the user can enable or disable the spooler. Disabling the spooler requires a reboot to make the change effective. However, the spooler can be started while the system is running, and take effect immediately.

The "Printers..." and "Queues..." items allow the user to change various aspects about the printer drivers and the queues they are attached to. They can also add and delete queues and printer drivers using these options. Each will bring up a dialog first asking which printer or queue is to be operated on. Once that has been determined,
a dialog specific to the printer or queue is displayed.

Printers

The first thing presented is a dialog representing the printer name chosen. Although not required, it is useful to reflect the type of printer in the name. There is also a short description field. The next field, labeled "Device" is the physical port to which this printer driver is connected. It is important when adding a new driver, or changing ports for an existing one that no other driver be connected to the same port. The Print Manager will not allow a conflict and the user will have to go to the driver connected to that port and disassociate it. The easiest way to do this is to select the "None" option for the port.

The transmission timeout is a setting for the Print Manager and spooler to determine how long to wait for the printer to respond before displaying a message that the printer is not responding. It is very important to set this value higher for PostScript® jobs than other jobs. Once data is in a PostScript printer, it will take longer than other printers. PostScript is an interpreted printer language and takes longer than most other jobs, thus a higher timeout is needed.

The other field in this dialog is the printer driver to be used with this printer name. The user is presented with a list of all drivers currently installed. If the needed driver is not installed, they will have to install it using the procedure described previously.

Once these fields have been filled in, the user may select the "Change" pushbutton, which will activate their changes, or select the "Printer Properties..." button, which will present them with another dialog. This dialog is presented by the selected printer driver. It contains information unique to that printer such as forms, paper orientation and fonts. Because of the wide variety of printers and options available, these details will not be covered in this article.

As with most dialogs, there are also buttons that allow the user to cancel the current operation or request help if needed.

Queues

The queue setup is similar to printers' setup. Each queue has a name and a description. Again, it is preferable, but not necessary to name the queue to indicate what its purpose or printer driver is. After the name and description is a queue driver name. OS/2 comes with two queue drivers, called PMPRINT and PMPLOT.

A queue driver is a program that removes an item from the queue and sends it to the printer. That is how the spooler takes the jobs and physically prints. The difference between PMPRINT and PMPLOT is that PMPLOT is designed to do *reverse clipping* often useful in plotted jobs. As such, PMPLOT is slower than PMPRINT.

PMPRINT is installed by default when OS/2 is loaded.

Next is a field titled "Separator". This indicates if there is to be a separator page associated with the queue. A separator page is a page that is printed prior to every print job that usually indicates the origin of the job, time and sequence number. Separator pages are usually used on network printers. If the user has a specially formatted page, its file name goes in this field.

The next two fields are for queue priority and scheduling. The priority is a number relative to all other queues running on that system. If all jobs are at the same priority level, they will run in the order they reach the spooler. Otherwise, the queue with the higher priority (lower number) will run before those with lower priority (higher numbers).

The scheduling field is used primarily with network servers. This function allows a queue to be scheduled to be active only certain times of the day. During times outside the schedule, the queue will be inactive.

Next is a list of printer names. These are the available printers to which you can attach this queue. Based on which printer is selected, the Printer Driver field is filled in.

Now the user can select "Change" to accept the changes, or go even further in the setup detail. By selecting the "Job Properties..." pushbutton, they can set up the specific properties for the jobs going to that queue. For example, they can set up two queues connected to one printer. By specifying different job properties, they can create flexibility in their applications.

For example, in a program which sometimes prints jobs in landscape orientation and sometimes in portrait orientation, rather than change the printer's default properties each time the application prints, the user can create two queues with nearly the same properties, except one is landscape and one is portrait. Then, instead of changing the printer's properties, the user can just send the job to the proper queue. *(Please note that this is dependent on how your particular application chooses to implement printing. Not all programs take advantage of this feature.)*

Once the user has finished with the queue setup, they can select "Change" and "OK" and their changes will be saved.

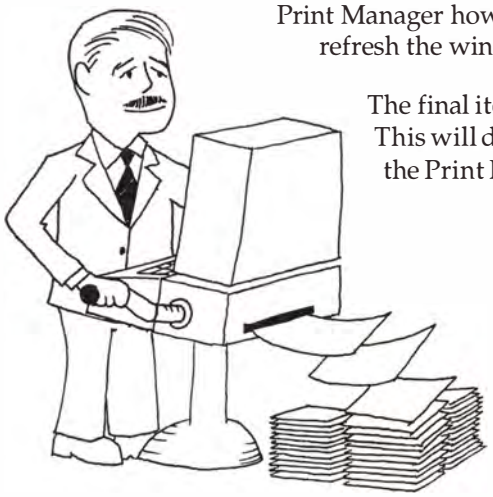
Application Defaults

The "Application Defaults" menu item allows the user to select which printer and queue they wish to use for defaults for their Presentation Manager applications. These will be used if their application either does not give them the option of which queue or printer to use, or it will be the initial selection displayed when they tell their application to print.

The final item is the DOS Timeout. This is used to let the spooler know how long to wait for print jobs from the DOS Compatibility environment before closing the spool file. DOS applications print very differently from OS/2 Presentation Manager programs and the user must tell the spooler how long to wait before printing the job.

The next menu is the "Refresh" menu. There are two options on this menu. The first tells the Print Manager to refresh the window immediately. The status of printers is very dynamic and must be periodically refreshed to see jobs enter and leave the queue as well as see the latest status of a printer.

The other menu item is used to specify an automatic refresh interval. This will tell the Print Manager how often to automatically refresh the window.



The final item is the "Help" menu. This will display help on any part of the Print Manager.

SUMMARY

You have seen an overview of the OS/2 Print Subsystem and how to use printing in OS/2. You've also seen the basics of how multitasking and device independence are handled and how to manipulate jobs and configure the various parts of this subsystem.

For a more in-depth discussion of the printing subsystem, please see the accompanying article in this issue.

David E. Reich, *IBM, Internal Zip 1424, 1000 NW 51st Street, Boca Raton FL 33429, is a senior associate programmer working with OS/2 Technical Support in Boca Raton. He has a MS in Computer Science from the State University of New York at Albany and has written several articles previously in the* ***IBM Personal Systems Developer*** *and has co-authored a book entitled* ***OS/2 Presentation Manager Programming****, published by John Wiley and Sons, Inc.*

Presentation Manager

The OS/2 Print Subsystem Architecture

by David E. Reich

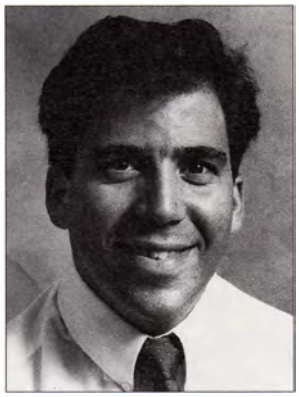
The Print Subsystem is one of the most powerful multitasking features of the OS/2® operating system. Along with this power goes a good deal of complexity. This article will explain how printing works under OS/2. The discussion includes the spooler, Print Manager™ and how the printer device drivers fit together to afford users and applications a wide range of printing functions.

Since OS/2 1.0 was introduced in December 1987, the printing subsystem has undergone significant changes. In OS/2 1.0, all applications were text-based. The original function simply managed output of several simultaneous applications to a device. Starting with version 1.1, Presentation Manager™ added the challenge of graphical hardcopy, and a new set of functions to printing under OS/2.

With a multitasking operating system, many programs can run at the same time. In such a system, it is not desirable to allow applications to print only while no other application is printing. However, it is not acceptable to allow several applications to print to a single physical device at the same time. If that were to happen, the output of different programs would be mixed together on the hardcopy device. The OS/2 print subsystem prevents this from happening.

The other primary function of the OS/2 print subsystem is to provide applications with device independence. In simple terms, this means the applications do not need to know the physical characteristics or capabilities of the output device: the operating system handles that. Device independence applies to OS/2 in general —the printer drivers provide independence for hardcopy devices in much the same way as the display drivers provide this for screen output — but it is especially important in printing.

There are three main components of the OS/2 Print Subsystem. They are the spooler, the printer drivers and the Print Manager. The function and purpose of each follows.



David E. Reich

SPOOLER

A spooler is not a new concept. The function of a spooler is a fairly simple idea; the implementation may not always be.

The OS/2 spooler is a program that intercepts output destined for a particular device or port on the computer. Depending upon the type of device and it's connection, this may be a parallel port (LPTx) or a serial (COMx) port. The spooler must be set up to handle each port it is manipulating, prior to use.

When an application sends output to a port, it cannot tell the difference between having the spooler intercept output and direct communication with the physical device. The spooler grabs the output before it gets to the port and places the print job in a file on disk. The spooler prioritizes the jobs and sends the saved files to the port which leads to the physical device. In doing this, the spooler ensures that only one job is being sent to a physical port at any given time.

The spooler queue processor is part of the spooler that is an important player, yet not all that complex. It removes jobs from the queue and sends them to the printer making sure that the job in the queue matches the printer

properties. If the job that is next in the queue has a mismatch, (for example if the job requires legal paper but letter is the current setting) the spooler will hold that job until the settings match.

Before OS/2 version 1.30.1, there was no easy way to find a mismatch. The only option was to look at the job details. Now, in versions 1.30.1 and higher, the status "Forms Mismatch" appears in the Print Manager window to indicate this situation.

There are two queue processors supplied with OS/2. They are PMPRINT.QPR and PMPLOT.QPR. The difference between the two is reverse clipping.

Reverse clipping describes a method of hidden line removal. As shown in Figure 1, without reverse clipping, lines may seem to intersect, where in reality they should overlap. This comes into play most prominently with plotters, where the order in which lines are drawn on the paper are important. On page-oriented devices, the page is built in memory and printed all at once. On vector devices, the lines are drawn individually and the order affects the final product.

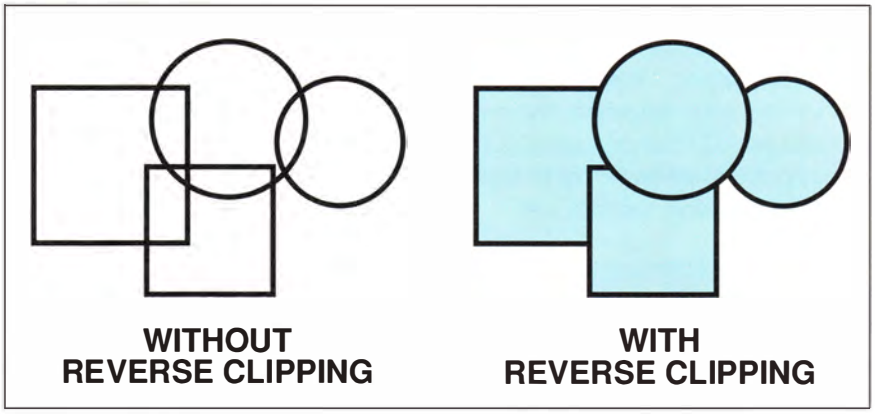


Figure 1. A Diagram Drawn With and Without Reverse Clipping

As one would expect, PMPLOT takes longer to render images on the printer because of the extended math to calculate reverse clipping. Either queue processor can be used on any device, but there are special cases when the user may wish to use one versus the other. Queue processors are defined on a per-queue basis, so one driver can be set up with two queues — one with PMPRINT and one with PMPLOT, depending on the type of job sent to the printer.

The spooler is the focal point of the print subsystem. The other components, including the Print Manager, center around the spooler, help it do it's job and help the user to interface with it.

PRINT MANAGER

The Print Manager is Presentation Manager's user interface to the entire print subsystem. It is also called the Visual Spooler Queue Manager.

The Print Manager shows the queues, both networked and local, and lets the user manipulate those queues. They can also view and change jobs currently in the queues and, beginning with OS/2 version 1.3, can install new printer drivers directly from the Print Manager. All these functions are the user's way of customizing the printing characteristics of their system. (A more detailed look at the Print Manager and how to use it can be found in "Printing Using OS/2" in this issue of *IBM® Personal Systems Developer*.)

INI FILES

Another major player in the printing process is the INI file. There are really two INI files, OS2.INI and OS2SYS.INI. Each contains some information about printing.

Most of the information contained in OS2.INI relates to program and system information. The OS2SYS.INI contains most of the information about printing. However, when writing programs to access the information contained in these files, programs simply call the Prf or the DosPrint function calls which handle that work. This will be discussed in detail in the next issue of the *IBM Personal Systems Developer*, in an article about programming printing functions.

Next we will discuss the flow of data from the time an application initiates a print job until the time the output appears on the paper.

PRINTER DRIVERS

Device independence with respect to printing is accomplished with the OS/2 Printer Drivers. As shown in Figure 2, the printer drivers perform a twofold function. The printer driver gets called upon the first time to assist in the creation of the spool file. Once the file has been spooled, the driver gets called again to actually print the file to the printer.

Every printer has it's own characteristics. To afford applications the freedom from knowing anything about the printer being addressed, the printer drivers contain device specific information relative to each printer.

Printer drivers work in a two-pass structure. In the first pass through the driver, a metafile is created by the print subsystem, assisted by the driver. As shown in Figure 2, the output from pass one is placed into the spool queue. So, files in the spool queue are metafiles. This fact can be very useful when programmers code their applications.

Included with OS/2 are picture facilities, such as the Display Picture Utility. This program can display metafiles on the screen. A preview of what the output will look like is obtained by holding the queue through the Print Manager, then looking at the .SPL file from the Display Picture Utility. The metafile shows what the file actually contains.

Pass two of the printer driver takes the metafile and actually does the printing. The spooler queue processor creates a presentation space, a device context and performs a GpiPlayMetafile to the printer. This is where the translation to the printer device takes place. The printer driver supplies the printer specific information here to create a data stream the printer understands.

A metafile is a much more compact way of storing the picture in the spooler queue than the raw printer stream. Since spooling allows the application to act asynchronously from the physical printing, the overhead to the system is negligible when going from the metafile to the printer data stream. Storing the raw data stream would be much more expensive in system resources.

The data path just described is the most generic path a print job will take. One might think of this as a *standard* data path. But there are many paths the print data can take from the application before reaching a printer. These are all determined by the application. It is up to the programmer which path best suits their application's needs.

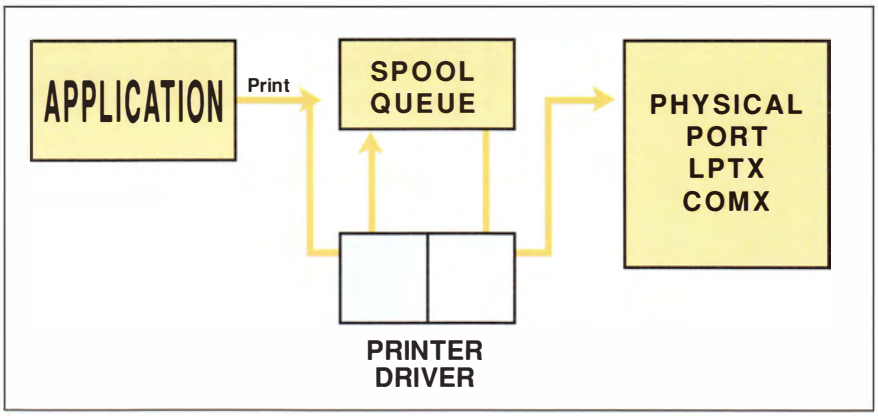


Figure 2. Data Path Overview

DATA PATH

As shown in Figure 3, there are several paths the print job can take, depending on how the application chooses to print. The two most common paths and their variants are direct versus queued, and standard versus raw printing.

Applications control how jobs are sent through the print subsystem. As shown in Figure 3, there are many options available to applications with respect to putting data on the hardcopy device.

Applications can bypass the spooler queues and have the output sent directly to the device. This does not defeat the spooling mechanism, but actually uses it to send jobs directly, thus preserving data integrity. The drawback in this case is that the application will still be busy printing, waiting for the device, whereas in a queued job, the job gets spooled to the queue, allowing the application to continue while the job prints asynchronously.

Programs can also send raw data to the printer, thus taking on the burden of knowing about the device, rather than letting the printer driver handle that work. This is often useful if

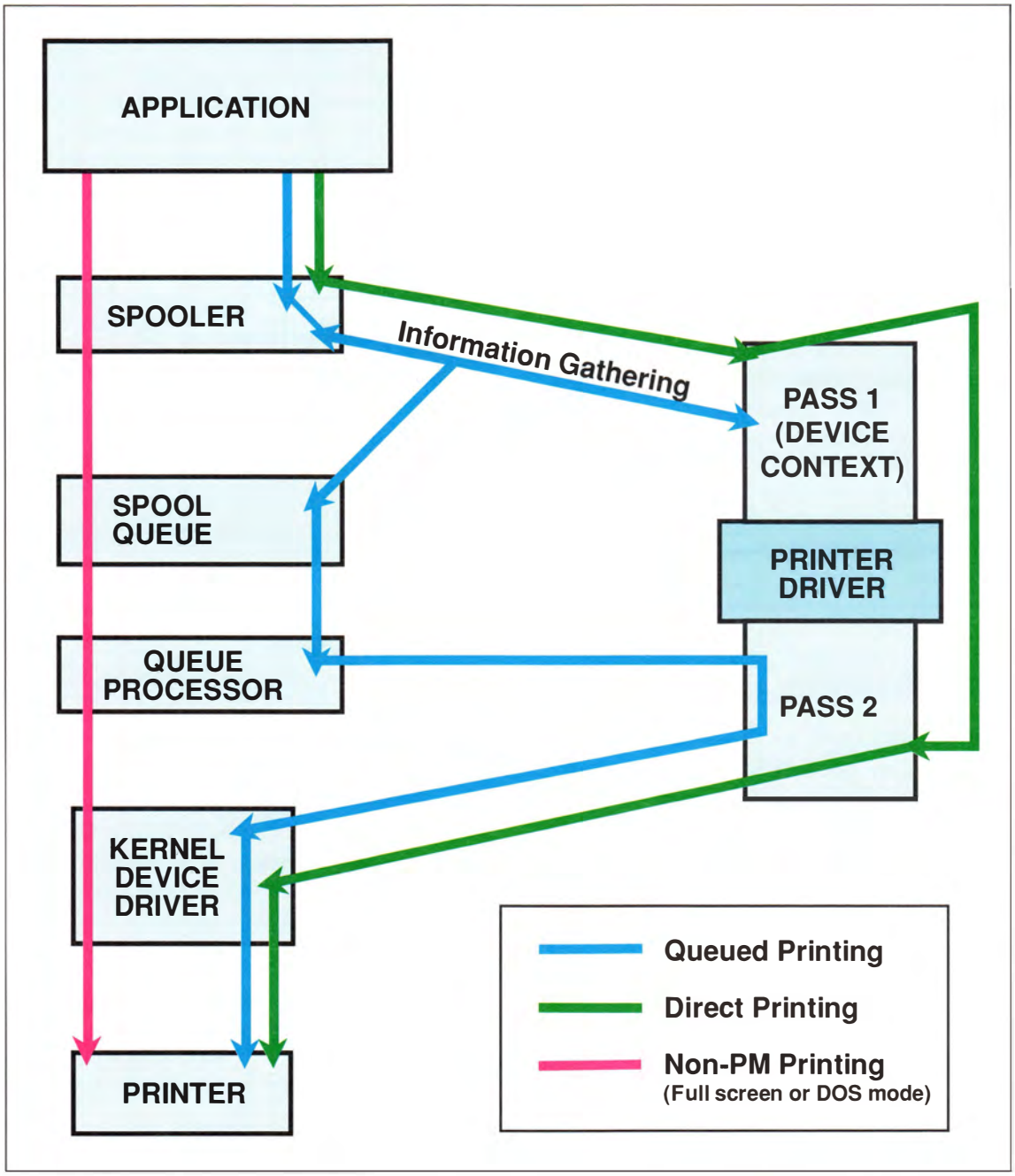


Figure 3. Data Path Details

an application developer wishes to take advantage of a printer or a feature of a printer not supported in the current printer driver releases.

When writing programs that print, the programmer must weigh all these drawbacks and advantages to decide which data paths they wish their print jobs to take. These items will be discussed later in this article and in more depth in the next issue of the *IBM Personal Systems Developer*.

STANDARD PRINTING

In standard queued printing, the application selects the data type of PM_Q_STD. This instructs the printer driver and the spooler to create a metafile and place it in the spool directory.

When the application opens a printer device context, the specification PM_Q_STD is seen by the spooler and it creates a metafile, using information provided by the application by way of the presentation space associated with the printer device context. The spooler queries information from the printer driver in this phase, which we have been calling *pass one*. The printer driver really does not do much here, other than provide requested information for the metafile creation. The resultant metafile is still fairly device independent at this point. Depending on what is in the print job, such as bitmaps and other graphics objects, this metafile can ordinarily be displayed by the Display Picture Utility.

At this point, the metafile is in the spool directory on the hard disk, along with a job file that describes characteristics such as paper orientation and whether the job was spooled as PM_Q_STD or PM_Q_RAW.

Now the queue processor comes into play. When it is this job's turn to be printed the queue processor opens the job and changes the Print Manager status to "Printing." The queue processor creates a presentation space, and a device context for the printer. It then drives the job through pass two of the device driver using a call to GpiPlayMetafile. The printer driver converts the graphics orders in the metafile to a data steam the printer understands. The data passes though the kernel device driver which controls the physical port, and out to the printer.

Although the process sounds complicated, it is actually fairly simple. In summary, the data moves from the application through the spool mechanism and using the printer driver, a metafile is created in the spool directory. When it's the job's turn to be printed, the queue processor takes the job from the queue and sends it through the printer driver to the port.

RAW PRINTING

Sometimes, it is desirable for the application to handle the creation of the raw data stream itself. Raw print jobs are a little simpler than queued jobs. They still follow the same data path, but instead of a metafile representing the job going into the spool directory, a file creating the raw printer data stream is created. In this case, the data passing through the printer driver is unmodified. The spool file is the actual printer stream that will be passed to the port, rather than a metafile to be played to the printer. This is signaled to the print subsystem by PM_Q_RAW in the job file.

The other main distinction regarding how print jobs are handled involves queued jobs versus direct print jobs.

QUEUED PRINTING

With queued printing, the print job is sent from the application through the printer driver (first pass) and into the spool queue. From there, the queue processor coordinates the removal of the job and sends it through the second pass of the printer driver and out to the physical device. This is specified by the program as OD_QUEUED.

As shown in Figure 3, queued jobs go from pass one of the printer drivers into the spool queue. This way, the application prints to the spool file and continues on. This is the most efficient form of printing because the spooler takes care of when the job is printed.

DIRECT PRINTING

When using direct printing, on the other hand, the job never goes into the spool directory. This is not necessarily raw printing, because the application can still specify standard printing to create the metafile as an intermediate step, but it is direct printing because the job does not get spooled. Instead, the job becomes a metafile and goes directly to the printer, bypassing the queue. In contrast to OD_QUEUED, this is specified as OD_DIRECT.

As stated previously, this does not defeat the spooling mechanism. Because the job is still going through the print subsystem and the kernel device driver for the port, the job is coordinated with others in the system (both other direct jobs and spooled jobs). However, a job printed OD_DIRECT causes the application to wait until the job is printed. Although the subsystem is still controlling the order in which jobs are printed, and

coordinating the execution among multiple simultaneous jobs, the thread of the application printing OD_DIRECT has to print the job itself. The queue processor takes no part in it.

PRINTER PROPERTIES

Much confusion has arisen from the terms job, queue and printer properties. All of these items have selectable options, and each set of properties serves a different purpose, but some of the items may appear to overlap. Due to how the dialogs interact with the user and each other, it is easy for an uninitiated user to become confused. This section will clear up these items.

Each printer has two classes of properties which vary depending on it's capabilities. First, there is the set of properties that require the user to take an action with the physical printer. These are properties such as using letter versus legal sized paper or serial versus parallel communications. These are the true printer properties.

The others are really not printer properties as such, but job property defaults. These are used to print a job if it is submitted without properties of it's own, (such as paper orientation, draft or letter quality and so on). These items are logical within the printer, and require no physical action.

JOB PROPERTIES

Job properties are those intangible properties about a print job that do not require the user to take an action with the printer such as feed in a sheet of paper manually, or change the paper tray. These items are stored on a per-queue basis to afford users the flexibility to send jobs with different properties to a single printer without having to change these properties in the system for each job.

It is important to note that it is entirely up to the application to take advantage of this function. Many applications do not yet do this. An application that supports job properties puts up a list of queues from which the user chooses. When a queue is selected, the application queries the INI file settings for the job properties associated with that queue and uses them in generating the print job.

For example, one might wish to have a PostScript® printer to which users can send portrait- as well as landscape-oriented jobs. As long as the application they are using supports this function, they can set up two queues, both associated with the PostScript printer, one with landscape defined in the job properties and one with portrait defined. When the job is sent to a queue, the properties associated with the jobs for that queue are used to create the job file and the printout comes out as desired.

When applications do not do this, they circumvent the function of allowing each job attached to a particular printer queue to have a set of job properties, and allowing the user to pick which queue to send the job to. They pull the job properties from those default printer properties described above.

QUEUE PROPERTIES

Queue properties vary on a queue-by-queue basis. These properties include specifications for separator pages and which queue processor is used. One could think of job properties as queue properties since they are stored per queue, but the distinction is made to illustrate that applications can manipulate job properties as they wish. They can use the default properties for the printer, pull the ones from the queue's job properties settings in the INI file or even take those settings and change them slightly if so desired.

LAN PRINTING

This article will not go into very much depth on LAN printing, but the basics will be covered here.

LAN printing is not much different from local printing. The main exception is that when the job goes from the queue, through pass two of the device driver and into the kernel device driver, rather than going to the output port, the LAN code intercepts it and sends it to the redirected machine. The job comes in on that

machine as if it were printed locally, and is printed as such. What happens on the workstation happens on the server.

To set up LAN printing, the workstation should be set up just as if the printer were local. The printer is then set up on the server locally. Next, the resource — not the port, but the queue associated with the printer — must be shared on the server. In the NET SHARE or NET ALIAS command, the /PRINT flag must be specified.

This creates a printer on a server, with a queue shared and a logical printer set up on the workstation. Assuming it is set up as LPT2 on the workstation, redirecting the LPT2 device to this shared resource will cause all output going to this printer to be sent over the LAN to the remote printer.

There are some important considerations when doing this. First, the printer driver on the server and the driver on the workstation must be the same driver. It makes no sense to have the IBM4019 driver on the workstation when the shared printer is a plotter. Another important item is that all fonts installed on the workstation must be on the server as well. If not, fonts used in applications will not be available to the server's printer and output will not be high quality.

Note that to hook up a printer driver to a port, the port must be defined in the OS2.INI file which comes set up with COM1-3 and LPT1-3.

But, for example, to install a printer driver on LPT4-9 on a LAN the programmer must add the ports to the INI file via a program. Figure 4 is a short program that can be used to add ports to the INI file.

```
/*************************************************************/
/* Add LPT4 to LPT9 to OS2SYS.INI                            */
/*                                                           */
/* INIUPLPT.c                                                */
/* Copyright IBM Corp. 1991, All Rights Reserved             */
/* After running this program, Print Manager will allow      */
/* you to connect                                            */
/* a printer to any one of the parallel ports LPT1 to LPT9.  */
/* LPT4 to LPT9 would have to be a redirected port, for example*/
/* from the LAN, as shown in the following sample command:   */
/*                                                           */
/* NET USE LPT9: \\LANSRV2\LANPRT5                           */
/*                                                           */
/* Author: Hans J. Goetz    January 1991                     */
/* IBM ITSC Boca Raton, Florida                              */
/*************************************************************/

#define INCL_DOSERRORS
#define INCL_DOSPROCESS
#define INCL_WINWINDOWMGR
#define INCL_WINSHELLDATA

#include <os2.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>

static   CHAR       ApplName[] = "PM_SPOOLER_PORT";
static   CHAR       ApplDefault[] = ";";
```

Figure 4. Adding Ports to the OS2.INI File (Continued)


```
void main()
{

   INT         i;
   CHAR        buffer[1];
   CHAR        KeyDefault[3];
   CHAR        *KeyName;
   CHAR        *KeyLoop;
   SHORT       rc;

   for(i=4;i<10;i++)                /* loop for LPT4 to LPT9     */
      {

      strcpy(buffer,"");            /* clear buffer              */
      strcpy(KeyDefault,"LPT");     /* initialize string         */

      KeyLoop = itoa(i,buffer,10);  /* convert integer to string */

      KeyName = strcat(KeyDefault, KeyLoop);
                                     /* concatenate strings      */

      /* write data to profile (OS2SYS.INI) */
      rc = PrfWriteProfileData(HINI_SYSTEMPROFILE,
                               ApplName,
                               KeyName,
                               ApplDefault,
                               (LONG)sizeof(ApplDefault));
      }
}
```

Figure 4. Adding Ports to the OS2.INI File

This was not done by default in OS/2 because not all workstations will be running LAN code. And even for those that are, if the LAN is not started, the extended ports will not be available anyway. The ports in the default INI file are those that can be physically placed into a PS/2. All others must be redirected, so it was decided to make this a user-modifiable feature.

The code in Figure 4 has been excerpted from the IBM ITSC Red Book, *OS/2 1.3 Volume 2: Print Subsystem*, GG24-3631 with permission.

DEPARTURES FROM BASICS: POSTSCRIPT

The PostScript data stream is really a programming language. As such, applications can not send a plain ASCII job to a PostScript device (unless, of course, the ASCII file is a PostScript program). Many printers, such as the IBM Proprinter® series and the Hewlett-Packard Laserjet® series can accept a plain ASCII file, but a PostScript printer only understands PostScript programs.

One of the most common mistakes made is copying a file, such as CONFIG.SYS to a PostScript® printer. The buffer light will flash and nothing will print. The buffer light indicates that data is being sent to the printer. But, because the data is not understood by the PostScript interpreter in the printer, nothing gets printed.

Another difference between PostScript printers and others, is that their jobs are always spooled as raw jobs. This is not to say that the applications must send the job PM_Q_RAW, instead it means that the file in the spool directory is not a metafile. This is the one exception noted earlier.

A job destined for a PostScript printer is in the spool directory as an ASCII file containing the PostScript program. This file can be viewed directly with a text editor (as opposed to using the Display Picture Utility as described earlier for metafiles). In this case, the first pass creates the PostScript file and the second pass is a straight pass-through step because the job is already generated in the printer's native data stream.

The PostScript driver has an added feature that allows the user to specify the job to print as a raw file or to an Encapsulated PostScript (EPS) file. These two are almost exactly the same, but have one significant difference: In printing a raw PostScript file, the user specifies a file name in the Printer Properties, Job Properties dialog. When a job is subsequently printed, the PostScript file is written to the file name specified. This is exactly the same file that would be in the spool directory. This file can then be copied to a PostScript printer later on because it is simply a PostScript program.

In contrast, an Encapsulated PostScript file cannot be printed. The purpose of such a file is to transfer images between systems. In an EPS file, the printer specific program instructions, such as paper feeds, are not present. Additionally, the commands that tell a printer to print the page (i.e. the PostScript showpage command) are also not present. There are many software packages that accept EPS files as input to merge parts of documents from different sources. The receiving application simply needs to imbed the EPS file into its document and send it down to the printer.

One other item of note with PostScript printers is that the PostScript language, by nature, is very slow. This is because the PostScript program is really just an ASCII file that must be sent to the printer and interpreted by the PostScript interpreter in the printer. As such, the printer buffer may fill up rather quickly, and the printer will not accept any more data until what has already been received is processed. This is especially true in lower memory (less than 2.5 megabyte) printers.

When this occurs, the printer will not respond to the computer until it is ready to receive more data. The computer may assume the printer is having problems such as a paper jam, because all it can determine is that the printer is not accepting data; no other information is available. In this case, the spooler will display a message stating that the printer is jammed or out of paper.

This problem can be avoided by raising the Printer Timeout Retry value in the Print Manager. This value tells the spooler how long to wait before putting up the jammed message. For most PostScript printers, a value around 300 or 400 seconds is about right. This does not slow down the printing at all, it simply tells the Print Manager to wait longer before assuming the printer is jammed. Note that the lower the amount of memory in the printer, the longer the timeout needs to be. This should help avoid frustration and *printer jammed* messages.

PLOTTERS

The plotter driver has a couple of items of note relating to performance. First, if the color sorting option of the driver (as available in the printer properties dialog) is used, it takes longer to process the plot file, but the physical plot will be faster. This is because each color is drawn completely before going on to another.

The other item was alluded to earlier. If PMPLOT is used in the queue processor, there are extensive arithmetic calculations that must take place to effect reverse clipping. This can add several seconds to minutes to plot jobs.

Both of these factors should be weighed when choosing a plotter setup.

OTHER MATTERS

This section will explain some of the miscellaneous items that don't warrant a section of their own.

Separator pages are not part of any print job, nor are they prepended to any existing print jobs. A separator page is a special job submitted by the spooler. This job will not appear in the queue, it will simply come out prior to any job in the queue it is associated with. Additionally, it cannot be used to set a

The Print Subsystem architecture will remain constant into release 2.0

printer into any special state. As a precursor to all jobs, the printer is reset to a default state.

However, a separator can be used to change emulation modes in a printer which supports more than one type of data stream. Let's say there is an IBM 4019™ printer attached to a system. This printer supports several data streams. Ordinarily, the user would have to reset the printer by switches. But, if several queues are set up, all of which are destined for the 4019, one queue can be set for each data stream, with the separator page file containing the control sequence to change the emulation mode of the printer to the desired stream.

Another item to note: printer driver setup can be tested by printing a file from the File Manager. This is done simply by dragging a plain text file, such as CONFIG.SYS from the File Manager to the Print Manager window using button two on the mouse. A prompt for which printer and mode (text or graphics) will appear, and the file will be printed (and can be viewed with the Display Picture utility.)

This leads to another point. In most cases, but not all, the Display Picture utility will be able to show a spooled metafile. There will be some cases where there is printer specific information in the metafile that the program will not be able to show. A good example is the PostScript driver. Since the spooled file is a PostScript program, it is not a viewable metafile. As printers become more complex, and more drivers are written by different vendors, previews through the Display Picture utility may not always be possible.

Full Screen programs do not use the printer drivers at all. They use the base kernel device driver to print. These applications cannot take advantage of the device independent functions of Presentation Manager. Jobs from these programs will simply be spooled along with other jobs in the system and are purely text-based.

One of the functions of a printer driver is to interact with the user by way of the Printer Properties dialog. It is the printer driver, not the Print Manager, that puts up this dialog. This is why each printer driver's Printer Properties dialog window is unique.

One other item of note is DOS-mode programs. These programs are able to print under OS/2, but just as with full screen programs, they cannot take advantage of the printer drivers. They go straight through to the kernel device driver and out to the printer.

SUMMARY

The OS/2 Print Subsystem is a complex mechanism. It is also very logical and straightforward. Once you understand the descriptions presented in this article, you should be able to perform almost any function in programming or any function in configuring or customizing printers and queues.

All of the topics presented here will continue to hold true in OS/2 2.0, the 32-bit version of the operating system. The printer drivers will continue to be enhanced and will remain compatible with the 16-bit version of OS/2. The overall OS/2 Print Subsystem architecture will remain the same.

Two more articles in this series covering programming printing and fonts, will appear in the next issue of the *Developer*.

David E. Reich, *IBM, Internal Zip 1424, 1000 NW 51st Street, Boca Raton FL 33429, is a senior associate programmer working with OS/2 Technical Support in Boca Raton. He has a MS in Computer Science from the State University of New York at Albany and has written several articles previously in the* ***Personal Systems Developer*** *and has co-authored a book entitled* ***OS/2 Presentation Manager Programming,*** *published by John Wiley and Sons, Inc.*

Presentation Manager

An Introduction to Multithreaded Programming

by Gary Murphy

Gary Murphy

WHAT IS MULTITHREADED PROGRAMMING?

Multithreaded programming support is a logical extension of multitasking. To understand what multiple threads are, one must first understand the concept of multitasking.

Multitasking is the ability of the operating system to switch between multiple processes running on a given processor, creating the illusion that programs are running concurrently. The ability of an operating system to multitask among several processes on the computer increases the total throughput of the system, since one task can execute while another is waiting for an event, such as disk I/O or user input, to complete.

While most mainframe operating systems, such as MVS and VM, support multitasking among applications, there is no concurrency within a single process. This is due to the absence of support within those operating systems for multiple threads of execution within a process. Therefore, the applications are designed so there is one thread — one single flow of logic — within the application.

Multithreaded programming techniques empower the applications developer to exploit the same parallelism within a given application that the operating system exploits among applications. Instead of executing single blocks of code sequentially, the application can start multiple code blocks for serial execution. This is multithreaded programming. The application can improve its throughput for the same reasons that the total system throughput is improved by multitasking.

Why Develop Multithreaded Programs?

There are several reasons for taking the time to learn to program an application with multiple threads of execution. A few are detailed below. As mentioned, a proper multithreaded design will increase the utilization of the computer resources, increase the throughput of the application, and potentially improve the usability of the application. It reduces, perhaps obviates, the need to confront the user of the application with the hourglass icon.

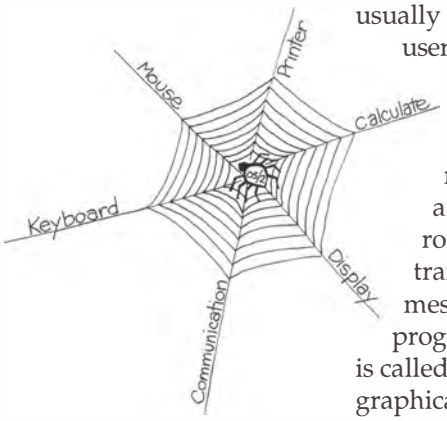
The Input Focus Problem

Multiple threads of execution offer a relatively easy solution to a programming problem that arises when programming for a graphical user interface, such as Presentation Manager™ or Microsoft® Windows.® OS/2® is a preemptively multitasking operating system. This means that if a thread of execution exceeds its allowed quantum of time (called a timeslice) on the processor, the operating system will suspend its execution temporarily and give the processor resource to another thread without the knowledge or consent of the application program. In spite of this, the developer of a Presentation Manager application still must develop the application in such a way that a given message is processed expeditiously and control is returned to the Presentation Manager subsystem promptly. The reason is a concept called *input focus*.

The rewards of multithreaded applications can be enjoyed by both developers and end users

In a Presentation Manager application, the currently active (highlighted) window is said to have input focus. When the user presses the mouse button to highlight another window, the window with current input focus is sent a WM_SETFOCUS message that must be processed by the window with current input focus before the other window will be highlighted. If the application window with input focus is busy processing another message, and takes a long time to do so, Presentation Manager applications will appear to *lock up*.

There are three programming solutions to the ubiquitous input focus problem. The first, usually unacceptable, solution is to present the user with the hourglass icon and force him or her to wait until the long-running routine is complete. The second solution is to break the single logical routine artificially into smaller pieces and implement the single logical routine as a sequence of *states* that transition by passing a user-defined message that requests a state change. This programming solution, implementing what is called a finite state machine, enables the graphical user interface manager, for example Windows or Presentation Manager, to get the WM_SETFOCUS message processed between state changes and allow the window focus to switch. These first two programming techniques are available for both Windows and Presentation Manager programs. The third alternative, available only to OS/2 programmers, is to execute the single logical routine as a separate thread.



More Intuitive Design

The discussion of the input focus programming problem enables an introduction into the next reason why multithreaded programming often is desirable. The application often can be designed in a more intuitive way. While decomposing a single logical function into a finite state machine is a workable solution to the aforementioned programming problem, it certainly is not the most intuitive. Multiple execution threads enable an application to wait passively for an event to occur instead of continually, actively polling to see if an event has taken place.

An example from personal experience is the design of an asynchronous communications program. In DOS, a typical design for a communications program might be to construct a polling loop that first checks for input from the modem, then checks to see if anything needs to be written to the modem, then checks for user commands. This tight loop keeps the program executing constantly, even if there is no I/O pending from the modem or the user. This design is CPU resource consumptive, but because of the single-tasking nature of DOS the computer is incapable of doing other processing, so the design is valid.

In a multitasking environment such as OS/2, the polling loop programming paradigm is replaced with a multithreaded solution. The design in OS/2 would implement a thread of execution to read from the modem and another thread of execution to write to the modem. If there is no data coming from the modem, the read thread will wait until there is data, much the same way that a typical DOS application waits for the user to type a command or press a key. Thus, the multithreaded solution is more likely to be consistent with prior programming idioms. It also makes better use of the computer resource, which often translates into better usability of the application.

Precursor to Client-Server Computing

Another reason to become familiar with multithreaded programming techniques is the industry-wide move to client-server computing. Simply stated, client-server computing involves a single application executing on two or more distinct computing platforms. While client-server computing is a completely separate issue from multithreaded programming, there are some conceptual similarities. If you think of the part of the application on the physical server machine as one thread of the application and the code on the physical client machine as another thread, the similarities begin to emerge.

As one example, an application now under development implements the client-server programming paradigm on a single computer. The requirements of the application suggested a design where one thread of the application

functions as a logical server with another thread performing the presentation services. This design would facilitate a future migration of the *server thread* to a separate physical server machine should the need arise.

INTRODUCTION OF NEW CONCEPTS

Multithreaded program design is desirable for almost all non-trivial applications. However, multiple threads of execution within an application introduce a new class of programming concepts and unique programming problems.

Interprocess Communications Facilities

The design of multithreaded applications require an understanding of the various interprocess communications (IPC) facilities that exist within OS/2. It is beyond the scope of this article to discuss these in any detail but a brief explanation follows to serve as a reference:

- **Semaphores.** A semaphore allows one thread to wait until execution of a second thread releases the first thread. This idea of waiting for a semaphore to clear is also called blocking on a semaphore.

- **Pipes.** A pipe allows a continuous stream of data to be passed between two threads. One thread writes information to the pipe, the other thread reads from the pipe. Each treats the pipe as though it were a file. UNIX® and DOS programmers may be familiar with pipes in the form of STDIN and STDOUT redirection using the "|" and ">" operators from the command line.

- **Queues.** Queues pass a packet of data between two threads using shared memory. Queues are not strictly limited to first-in first-out (FIFO) ordering.

- **Presentation Manager messages.** User-defined Presentation Manager messages are usually not sent between separate processes, so they usually are not included in the list of interprocess communications facilities. However, for communication of information between threads of execution in a single process, they can be very effective tools.

Multithreaded applications introduce some degree of parallelism. Conceptually, this means that two (or more) parts of the program are executing simultaneously. In reality, the operating system creates this illusion by switching frequently among the threads in this (and other) applications. What this means to the programmer is that a thread may be interrupted between any two machine instructions to enable another thread of executing code to access the CPU resource for a short duration.

Critical Sections

There may be times during the execution of a thread that it must complete a series of instructions without the possibility of another thread within the application being dispatched. One example is the case where a data structure is shared between two threads. During the course of execution one thread may need to update multiple fields in the structure. It is essential that the other thread is not allowed to execute while the fields are being updated, since until all fields are updated with the new values, the structure is in an unstable state. The lines of code that must execute as a single entity without possibility of interruption are called a critical section.

OS/2 provides a simple and effective means of assuring that a critical section of code is executed without the possibility of another thread within the application being dispatched on the processor. The applications programming interface calls (APIs) DosEnterCritSec() and DosExitCritSec() bound the lines of code that must execute as a single entity. Since a critical section suspends execution of all other threads in the application, it is important to minimize the time spent in a critical section.


```
/**
***  Thread 1 is a standard Presentation Manager message
***  handling routine.  When it receives a timer message, it
***  checks to see if there is any new information to present
***  to the user.
***
***  "pServer" is pointer to a structure that is shared between
***  the two threads.
**/
   .
   .
   .
case WM_TIMER:
   DosEnterCritSec();
   if (TRUE == pServer->fUpdated)
      strcpy (pszString, pServer->pszText);
   pServer->fUpdated = FALSE;
   DosExitCritSec();
   .
   .
   .
   break;

/**
***  Thread 2 implements a service provider that passes character
***  strings to another thread providing presentation services
***
***  "pServer" is pointer to a structure that is shared between
***  the two threads
**/
   .
   .
   .
Dos EnterCritSec();
strcpy(pServer->pszText, pszNewData);
pServer->fUpdated = TRUE;
DosExitCritSec();
   .
   .
   .
```

Figure 1. Using Critical Sections to Control Access to Information Shared Among Multiple Threads

Deadlock

A deadlock situation occurs when two threads are waiting on each other. Assume the following: One thread calls the DosEnterCritSec() API, then issues DosSemSetWait(...) to set a semaphore and wait for another thread to clear the semaphore. This thread is in the midst of a critical section, which means that no other threads in this process can execute until the critical section is exited. It then blocks on a semaphore and waits for another thread to clear the semaphore so it can continue. Clearly, this *Catch-22* situation would never be resolved and the entire application would be deadlocked. That is, it would wait forever for a set of circumstances that never can happen.

Unfortunately, most deadlock situations are not this obvious. They can be exceedingly difficult to debug, since the only symptom is


```
/**
***  This thread will cause the deadlock of the application.
**/

DosEnterCritSec();
   .
   .
   .
   .
DosSemSetWait(&hsem, SEM_INDEFINITE_WAIT);
/* This code will never be executed */
   .
   .
   .
DosExitCritSec();

/**
***  This thread never gets the opportunity to clear the semaphore
***  after it is set by the above thread since the above thread
***  is in the middle of a critical section.
**/

   .
   .
   .
DosSemClear(&hsem);
   .
   .
   .
```

Figure 2. An Illustration of a Deadlock Situation Between Two Threads

that nothing happens. The best way to avoid this bug is careful design and attention to detail when coding.

Thread Serialization and Synchronization

Unless the execution of threads are synchronized explicitly using an IPC facility such as semaphores, the developer is unable to determine the point of execution in one thread relative to another thread in the process.

If two threads update a common variable, it is possible that the results of the execution of the application will be different depending on which thread updated the variable first. Furthermore, repeated executions of the application may return different results depending on the order in which the threads were dispatched by the OS/2 operating system.

Again, this kind of bug is difficult to detect since the outcome can not be recreated on demand, and the use of debuggers is likely to alter the execution sequence of the threads. These situations can be avoided, once again, by designing the application to avoid such situations. The correct use of semaphores and critical sections can avoid this type of bug.

DESIGN TECHNIQUES

Introduction of multiple threads can ease applications design in many cases. Multithreaded design also will solve a certain class of programming problems. However, it also introduces a new set of programming concepts and programming problems such as those listed above. The best defense against defects is a fundamentally sound design. Every application design is different, but certain rules-of-thumb can reduce the

likelihood of code defects introduced by multithreaded design.

Thread Autonomy

Many times the application requirements will allow a thread to perform a functionally isolated unit of work. By isolated, it is meant that the thread relies very little on other variables in the application.

For example, an application may need to read a file and fill a data structure based on the file contents. An acceptable design would create a thread of execution that allocates a structure, reads the file, parses the information, and fills the contents of the data structure. When complete, the thread could post a user-defined Presentation Manager message to the main thread with a pointer to the filled data structure. This thread is isolated in that it relies on little information (except perhaps a copy of the file name to read) from the main thread. There is no need to communicate the progress of the I/O operation to the main thread. From the main thread's point of view, it has asked for file information and dispatched a unit of work to get it. It doesn't care what happens until the event (reading the file) has completed.

Data Ownership

It sometimes can be beneficial to think of one of the threads as the *owner* of the variable or structure. This implementation would allow only the owner to modify the contents of the

```
/**
***  Thread 1 is a standard Presentation Manager message
***  handling routine.  When it receives a user-defined message
***  with new information, it presents it to the user
***
***  In this example, "mp1" points to the new text to display.
**/
   .
   .
   .
case UWM_NEWINFO:
   .
   .
   .
   strcpy(pszString,PVOIDFROMMP(mp1));
   free(PVOIDFROMMP(mp1));
   .
   .
   .
   return;
/**
***  Thread 2 implements a service provider that passes character
***  strings to another thread providing presentation services.
**/
   .
   .
   .
strdup(pszCopy, pszNewData);
WinPostMsg(hwnd, UWM_NEWINFO, MPFROMP(pszCopy), NULL);
   .
   .
   .
```

Figure 3. Using Presentation Manager to Pass Information Between Two Threads in the Same Process

datum. Other threads are allowed only to read the contents. This design probably would require a critical section in the owning thread while the datum is being updated.

The idea of a thread owning the data can be extended to implement code that borrows heavily from object-oriented programming. Instead of sharing data between threads, the owning thread may encapsulate the data within its thread and pass copies of the data to threads that request the information via the Presentation Manager WinSendMsg API. While this design is not as efficient in terms of computer resources as some other design techniques, it is easier to implement and thus reduces the likelihood of introducing a timing-dependent error.

Guarantee Unblocking

When one thread blocks on a semaphore, it is essential that another thread is guaranteed to unblock it. If the code that unblocks a semaphore (e.g., DosSemClear(...)) is in an if statement, the potential exists for the thread to be blocked forever.

Similarly, your design must assure that the code to clear the semaphore hasn't already executed before the code is executed to wait on the semaphore. This is called a race condition since the code that blocks on semaphore is in a race with the code that clears the semaphore. If the thread that clears the semaphore wins the race, your code is the loser. Another analogy for this condition would be waiting for the train that has already left the station. Your code might wait forever.

Code Synchronization

One method of synchronizing the execution of the threads that avoids race conditions follows:

First, set the semaphore (e.g., DosSemSet(...)), then start the thread that ultimately will clear the semaphore (via DosSemClear(...)). The initial thread continues to perform some work in parallel with the newly started thread. Eventually, it will request to block on the semaphore (e.g., DosSemWait(...)).

If the second thread happens to clear the semaphore before the initiating thread blocks on it, no harm is done. The initial thread will continue to execute without stopping. If the second thread has not yet reached the code that clears the semaphore, the initial thread will block on the semaphore and wait for it to do so.

SUMMARY

The rewards of developing multithreaded applications can be enjoyed by applications designers, developers and end-users. However, these rewards are obtained only when the data processing professional understands the implications of multiple threads of execution and then designs and codes applications to use them effectively. Not only will the applications make better use of the computing resources, but they also will make better use of the more important *end-user resource* by reducing or obviating the dreaded hourglass icon.

In mastering this area, the applications programmer also develops a skill set that will be of great value in what many consider to be the pervasive programming paradigm for the next decade: client-server computing. While acquiring the skill set necessary to implement multithreaded applications is not a trivial process, the benefits of having made the journey are well worth the effort.

Gary Murphy, *is a senior programmer for Hilbert Computing, a consulting and contracting company that specializes in OS/2 PM applications. He received his BS in Computer Science from Iowa State University in 1982, and was an MVS and VM systems programmer until 1990, when he began developing vertical market applications using OS/2. Mr. Murphy can be reached via CompuServe® at [73457,365] or on Prodigy at [VWSD07A].*

Presentation Manager

Split-Line Multiple Session Communications

Ken Bain

by Ken Bain

INTRODUCTION

Communication software always has been one of the core applications developed and used on microcomputers. Such software has many different uses. Among these are providing the user with the capability to copy data and programs from one computer to another, and the ability to call bulletin boards for information and data sharing. Communication software provides a fast, effective, and inexpensive way to exchange data and information. The needs of both the data processing professional and the computer industry, of the 90's and beyond, dictate that communication software will be a requirement for almost every desktop computer in the near future. In fact, the demand for communication software will continue to grow as many more homes utilize computers for family entertainment, and its use in the business sector will continue to expand, involving such applications as real-time banking and online marketing. With the introduction by IBM® of OS/2®, with its ability to run multiple processes, the demand for communication capabilities under this powerful operating environment has increased significantly.

Most communication processes under DOS monopolize the entire system's resources, thereby forcing the user to wait until the process is completed. For example, if a user is transmitting a file using a DOS communications product, they must wait for the transmission to complete before being able to begin other tasks (e.g., work on spreadsheets, do word processing, etc.). A possible exception occurs in the case that the DOS communication product being run is Terminate and Stay Resident (TSR), and the communications program does not use all of the available memory. In this situation the user may be able to run another process simultaneously. However, it is likely that the only processes that could run concurrently with the communications software are those that have small memory requirements and do not contain a lot of interrupts .

Communications under Windows® works well, but only with a single communication session running. Unfortunately, if another session is started which requires a lot of hard disk access (e.g., a spreadsheet), the computer is restricted to a certain way of constructing the system's work environment, and resources again are being wasted. Communication software for Windows is limited inherently by the one-dimensional nature of the DOS platform.

In contrast, communication capabilities under OS/2 are restricted only by the number of communications ports on the computer, and are limited temporarily only by the speed at which the port can be set. Under OS/2, a program can have multiple preempting threads set, giving the user the ability to execute tasks with a higher priority before completing other tasks of less importance. For example, a thread could be set to read the data coming across the COM port while another thread is reading characters being typed at the keyboard. Also, OS/2 allows smooth and efficient memory sharing with the Clipboard and Dynamic Data Exchange (DDE). You can cut and paste between programs, and send data to, and receive data from applications running on the same desktop, all automatically.

SPLIT-LINE

Split-Line™ is an asynchronous communication product which is Common User Access™ (CUA™) and Systems Application Architecture™ (SAA™) compliant, and is designed with emphasis on the power of OS/2's multi-thread and shared memory capabilities. Split-Line contains the same elements as most other communications packages, but also contains many enhanced features. These features include: programmable function keys with pull-down menus, a scrollable 202-line buffer, DDE (referred to as Remote Links in Split-Line), clipboard, and the ability to assign modems to a COM port vs. assigning them to a user or assigning the COM port at the time of the call.

Split-Line was designed to take full advantage of the OS/2 Presentation Manager™ operating system environment. Installation is automatic and in PM format, with simple-to-follow instructions. Each Split-Line window utilizes OS/2's sizing, moving and other capabilities. The product conforms to the CUA and SAA standards (1990 as defined by IBM), and supports the IPF help function.

You can run as many Split-Line sessions as you have COM ports on your computer. For example, your system could be receiving a file in a first Split-Line session, sending a file in a second session, and communicating with a bulletin board in a third session. All of these sessions could be running simultaneously while you are running your spreadsheet and word processing software in other OS/2 windows.

Split-Line has 24 user programmable keys. The first 12 User Defined Keys (UDK's) are displayed in the user menu when Split-Line is online. The information bars and function key icons can be displayed or not at the user's option. Information bars contain the following items: Time, Date, Time online, Key and Line Status, Baud Rate, Data, Stop Bits, Parity, Protocol and Capture Status (see Figure 1). You can transmit a character simply by double clicking on the character, or transmit a highlighted word or phrase to the host by using the Paste-to-Host option. Split-Line also supports popular file transfer protocols. You can capture a conversation to a disk file and can review the 202-line buffer. You also can print the screen, the entire 202-line buffer or the clipboard. (See Figure 2.)

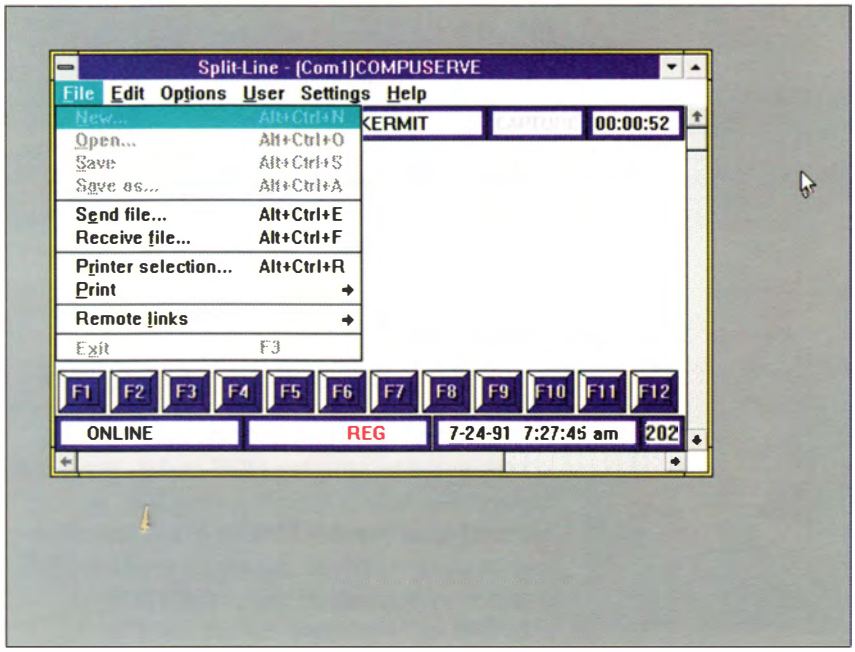


Figure 1. Split-Line is a Versatile Terminal Emulator and Communications Program. It is CUA and SAA Compliant, and Supports DDE Both as a Server and as a Client.

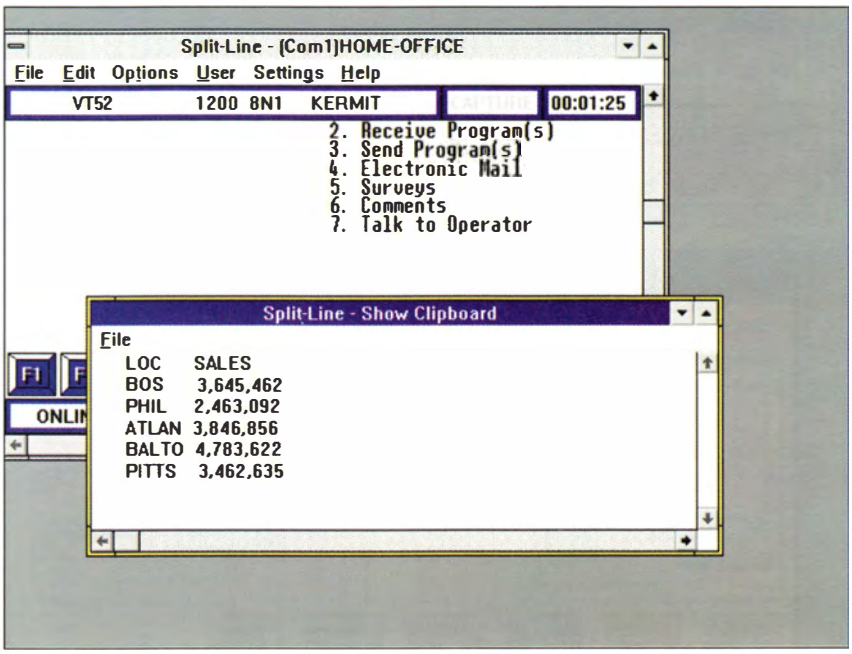


Figure 2. Split-Line Allows Screen Text to be Highlighted and Copied to the Clipboard. The Clipboard Contents can be Pasted to Other Applications, a File, a Printer, or the Host.

USING THE CLIPBOARD IN SPLIT-LINE

Split-Line uses the clipboard in the CF_TEXT mode. You may copy text from any session on your desktop. Then, with Split-Line you can view that information, paste it to the Split-Line buffer or print the data. Additionally, you may take the information in the clipboard and paste it to a file for retrieval at a later time, or paste the information to the host computer. This is one of Split-Line's greatest advantages. Split-Line also has the ability to print the entire 202 line buffer, and you can copy the entire Split-Line buffer to the clipboard so that it may be pasted into other sessions.

USING DYNAMIC DATA EXCHANGE (DDE) IN SPLIT-LINE

A large part of the development of Split-Line was devoted to utilizing DDE and its powerful capabilities fully. (In Split-Line, DDE is referred to as Remote Links.) We discovered that in many applications which support DDE, it was very difficult to understand how to establish and terminate Server and Client links. When the DDE (Remote Links) modules of Split-Line were being developed, it became apparent that not all applications have a standard way of handling Remote Links or DDE signal information. Thus, it was necessary to consider how different applications handle DDE signals. Also, Remote Links had to take into account the facts that terminology is different among applications, the lengths of topic and item names are not standard, some applications allow special characters, and some applications are case sensitive.

Split-Line handles the speed of receiving data, the data being broken down, and the speed of the system transferring DDE from Split-Line to another application with a special queue. That way, if Excel® or Lotus® cannot accept data because it is updating a spreadsheet, the data coming across the line is not lost. Taking these facts into consideration in the development of Split-Line, considerable efforts were made to make the Remote Links portion of Split-Line as user-friendly as possible. When using Split-Line, you can build a file which contains the Remote Links information specified for each application, so that later you can assign this file to a phone entry. As many as 40 Remote Links can be defined in Split-Line. Furthermore, Split-Line gives you the ability to see all programs which are supporting DDE topics on the desktop.

Client Items

Split-Line can conduct several DDE conversations at the same time, and can do so both as a server and as a client. Client topics are displayed in a list box. You simply can double-click on the application topic combination that is active and then add the item name. For example in Excel the name would be a cell or a range of cells (Example RC or A1); in Describe® it would be a tag name. Upon receiving a client item from the server, Split-Line examines the data received and if the data does not contain a CR/LF, Split-Line appends one and then transmits the data to the host computer. Depending on your application, the host then could update a corporate spreadsheet or report automatically. (See Figure 3.)

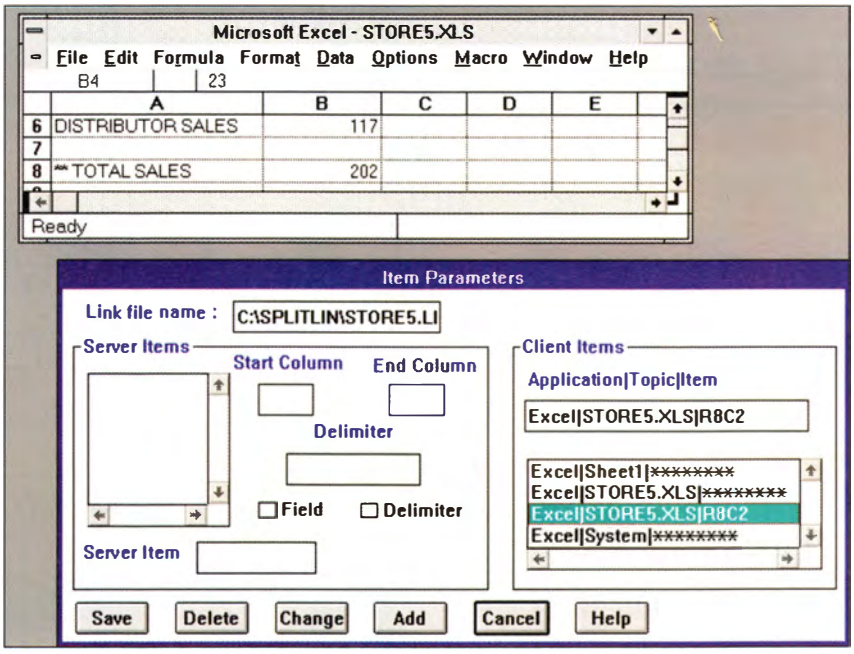


Figure 3. Split-Line Supports DDE as either a Server and/or a Client. Several DDE Conversations Can Occur at the Same Time…as a Client Application in Some and as a Server Application in others.

Server Items

Split-Line takes a different approach in applying DDE to incoming data. With Split-Line, you have the ability to take an incoming line of data which is up to 132 characters in length (and terminated by a CR/LF), and delimit the line into as many as 40 different segments which can be assigned item names. For example, a line of incoming data from the host computer could contain the elements quantity, price, on hand balance and Y-T-D information. If these items are always in the same columns, you could column delimit the line and send each server item to a different program. In a different scenario, you could have an incoming line of data from the host which is delimited by characters (e.g., comma, space, etc.) and then assign that delimiter an item name. Each time that character is

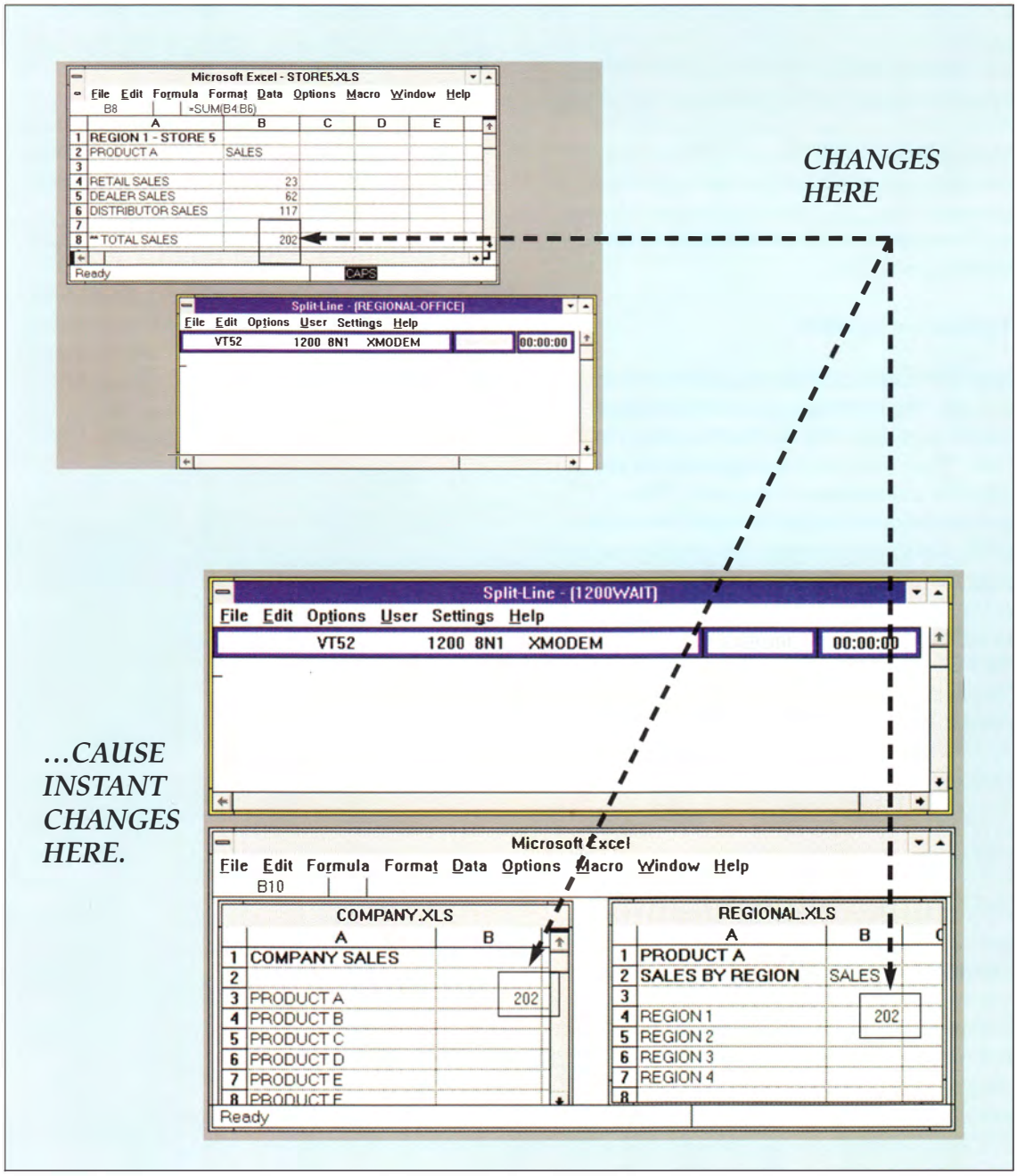


Figure 4. User's View of Dynamic Data Exchange Using Split-Line

received, the information bound by the character delimiter would be served to whichever applications are linked to Split-Line.

DDE Example

To exchange data, the participating applications first must engage in a DDE conversation. The server application must be running first; however, it is the client application that initiates the conversation. The application that initiates the conversation is known as the client and the application that responds to the client is known as the server. After a conversation has been started, the client interacts with the server by issuing messages which ask the server to perform given actions. The client application always receives data and the server application always sends data.

Application Example

Split-Line can act either as a client or a server, or both. Figure 4 shows the two computers communicating with each other using Split-Line. They also are running a second session which is a spreadsheet program. The spreadsheets are linked to Split-Line using DDE. On the top screen, the spreadsheet is the server and is sending data to Split-Line (client) as the data changes. When Split-Line, acting as a client, receives data, it sends the data out the COM port. On the bottom screen, Split-Line is the server and the spreadsheets are the clients. As Split-Line receives the data sent by the other computer, it sends the data directly to the spreadsheets (clients).

CONCLUSION

While the basic communication processes among computers are not likely to change, communication methodology and efficiency can improve significantly under OS/2. Communication can be a background process, and one PC can communicate with several others, while running multiple applications simultaneously, and while sharing common data via DDE. In an attended mode, this new way of communicating will enable use of a mouse to cut and paste from one application to another over a phone line. After a user experiences the power and functionality of communications under OS/2, it is unlikely he or she will be satisfied with the old ways under DOS ever again.

Ken Bain, *OSI Software, Inc., 8111 Timberlodge Trail, Dayton OH 45458. Mr. Bain is the Vice-President of Research & Development at OSI Software, Inc. and holds degrees in Systems Analysis, Computer Programming, and Electronics. While serving in the USMC he worked with IBM mainframes. Later he worked in data communications with the Montgomery Ward Distribution Center and various other companies, on mainframe, micro- and mini-computers, interfacing unlike systems, and handling data conversions. He has held various positions including Data Processing Manager, Director of MIS, Programming Manager, and Vice President of Research and Development, and has worked in the computer industry since 1972. He can be reached at the above address, by phone at 513-434-8668, or by fax at 513-434-5962.*

Communications Manager

ISDN: What it Can Do Today

By Katie Barrett, Bill Bulko, Frank Grubbs, Steve Kenney, Denise Loose, Pratik Nanavati and Pat Scherer.

We are all members of the OS/2® Communications Manager™ development effort. Recently we got together to answer some general questions about Integrated Services Digital Network (ISDN), the same questions that we had when learning about ISDN. The discussion that follows begins with the definition of ISDN and proceeds to applications and cost benefits of ISDN.

This paper is an introduction for future articles which will provide product specific information for ISDN applications on OS/2.

WHAT IS ISDN?

ISDN is a technology for simultaneously transferring voice and data over normal digital telephone lines providing a fast and inexpensive way to share information. ISDN line usage is typically charged by connection time just like voice calls. The digital phone line that ISDN uses is logically organized into three channels: two B channels and one D channel. Each channel can support a separate voice or data stream, somewhat like having three simultaneous telephone calls going through one telephone. Although ISDN channels can be used in other combinations, this 2B+D channel configuration, called *Basic Rate*, is the most common.

ISDN is a CCITT standard for digital switched public telephone connections. The terminal equipment at each end of the connection can exchange information, either voice or data, on a call-by-call basis.

The CCITT ISDN standard for Basic support covers three major areas:

1. PHYSICAL HARDWARE, ELECTRICAL INTERFACE, AND WIRING.

The CCITT standard defines a 4-wire, full duplex, RJ11 connection for wiring, i.e., *plain old telephone* wires, and a time division multiplex for the electrical interface. The clocking rate is 192,000 bits per second (192 Kbps). The multiplexing divides the 192 Kbps into:

- A 16 Kbps signaling channel (D channel),
- Two 64 Kbps channels for information (B channels),
- And 48 Kbps reserved for clock timing and maintenance.

This standardizes the configuration of the electrical interface between the terminal equipment and the switch, which, in the past, differed with each manufacturer.

ISDN promises a global interconnection of diverse hardware and software

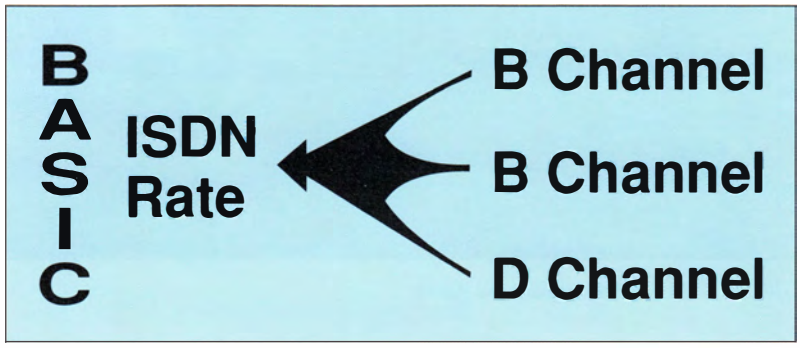


Figure 1. Basic Rate Channels

2. D-CHANNEL SIGNALING

The D channel is used by the terminal equipment, the *telephone*, and the switch, the *telephone network* like AT&T® telephone network, to place an outgoing telephone call, receive an incoming call, and disconnect a completed call. Since ISDN is a synchronous digital network, all D-channel information must be in a CCITT standard message format. These formats define all functions dealing with the physical connection of two pieces of terminal equipment through a switched telephone network. *Terminal equipment* refers to any end user interface such as a telephone, workstation, minicomputer or mainframe.

This standardization provides for a worldwide *open system* for the switched telephone network replacing the private signalling protocols supplied by each manufacturer of terminal equipment in the past. Therefore, equipment provided by any terminal vendor can connect to any other vendor's ISDN network.

Authors L-R: Bill Bulko, Pat Scherer and Katie Barrett

3. B-CHANNEL

The data rate of each B-channel reflects the standard rate for digitizing voice on a network (64 Kbps). The ISDN CCITT standard for the B channel is for *clear channel use* meaning that no restriction is imposed by the network on the content of information exchanged between two pieces of terminal equipment with a physical connection established through D channel signaling. In other words, data can be transmitted in any communications protocol that can be transmitted digitally and can be understood by both the sending and receiving terminal equipment.

The *clear channel* standard provides the maximum flexibility for terminal equipment vendors to provide function to the end user. For example, a personal computer with an ISDN adapter card could be used to place a call to a SNA Host system and connect a 3270 SNA emulator over ISDN to the host. At the same time, the user could place a call to a facsimile machine and exchange FAX sheets on the other B channel. When the FAX job is completed, the user could place a voice call home, using the normal handset to talk and listen.

HOW DOES ISDN COMPARE TO THE TYPICAL LOCAL OR WIDE AREA NETWORK?

Local Area Networks (LANs) can transmit data at rates exceeding 10 Mbps (megabits per second), although commercially available LANs today operate at 4, 10, and 16 Mbps. The trade-off for this speed is that LANs can only be used to communicate over limited distances. Wide Area Networks (WANs), on the other hand, are not limited by distance but are typified by a much slower data rate of between 1.2 and 64 Kbps. A WAN may be a connection to a mainframe host made by way of an asynchronous modem, or a complex *enterprise* network interconnecting several LANs to a remote host through use of special wiring and LAN gateways.

Functionally, ISDN's support of simultaneous voice and data communications is a feature never before exploited by commercial LANs or WANs. Use of ISDN is not limited by distance

as are LANs. ISDN offers a basic data rate of 144 Kpbs which is equal to or better than that of any other commercial WAN. ISDN also supports a primary rate of 1.544 Mbps (US Standard) and 2.048 Mbps (European Standard); thus approaching speeds only available before with distance — restricted LANs.

WHAT ARE NEAR TERM APPLICATIONS FOR ISDN?

A number of exciting applications are being used and developed for ISDN. Among them are products which use the capability of identifying the source of an incoming call to access customer account information on the caller. For example, when Mrs. Jones calls her bank to inquire about her account, the bank teller answering the call automatically sees her account information on the computer screen and answers, "Hello, Mrs. Jones, how can I help you?" As the teller no longer has to acquire Ms. Jones account number and manually key it in to access the account information, this represents a substantial improvement in customer service. In addition, the account information allows the teller to greet Ms. Jones in a more personable way.

Another application is teleconferencing. Information can be sent over a data channel and simultaneously discussed over the phone. A person could, for example, get help on an income tax return by calling the accountant, sending a softcopy of the return, and then discussing the portions which both parties have displayed on the computer. Merchandising applications may use this feature to send images of their products to be viewed by a potential customer while either a salesperson or voice application describes the product and takes information for the order. An application like this could be used to go house-hunting, Christmas shopping, or maybe just *window shopping* without ever leaving your home.

Other applications simply take advantage of the higher data rate and ease of installation. One of the most common uses is to connect two or more remote Local Area Networks (LANs). ISDN, while not as fast as a LAN, allows wide area networking at speeds equal or greater to those supported by most permanent connections. This is a considerable improvement over asynchronous communications.

The fact that ISDN does not require special wiring, and is relatively easy and economical to install, makes ISDN a good choice for use as an auxiliary communications line to improve network performance during peak loads. ISDN may also be used as a backup to enable continued service when problems are detected on the primary networks.

The portability of ISDN makes it ideal for accessing data from remote locations allowing, for example, bank loans to be made on-the-spot at trade shows, or a salesman to plug into a telephone line with a laptop and exchange sales data with the home office.

The fact that ISDN uses the same equipment that is found in almost every home and business significantly expands the possible applications. Anyone with telephone service is

Authors L-R: Steve Kenney, Denise Loose and Pratik Nanavati

wired for ISDN, provided that their local telephone switch supports ISDN. Add a computer and ISDN adapter and that person has all the components needed to simultaneously send data and voice to any other telephone connection in the world.

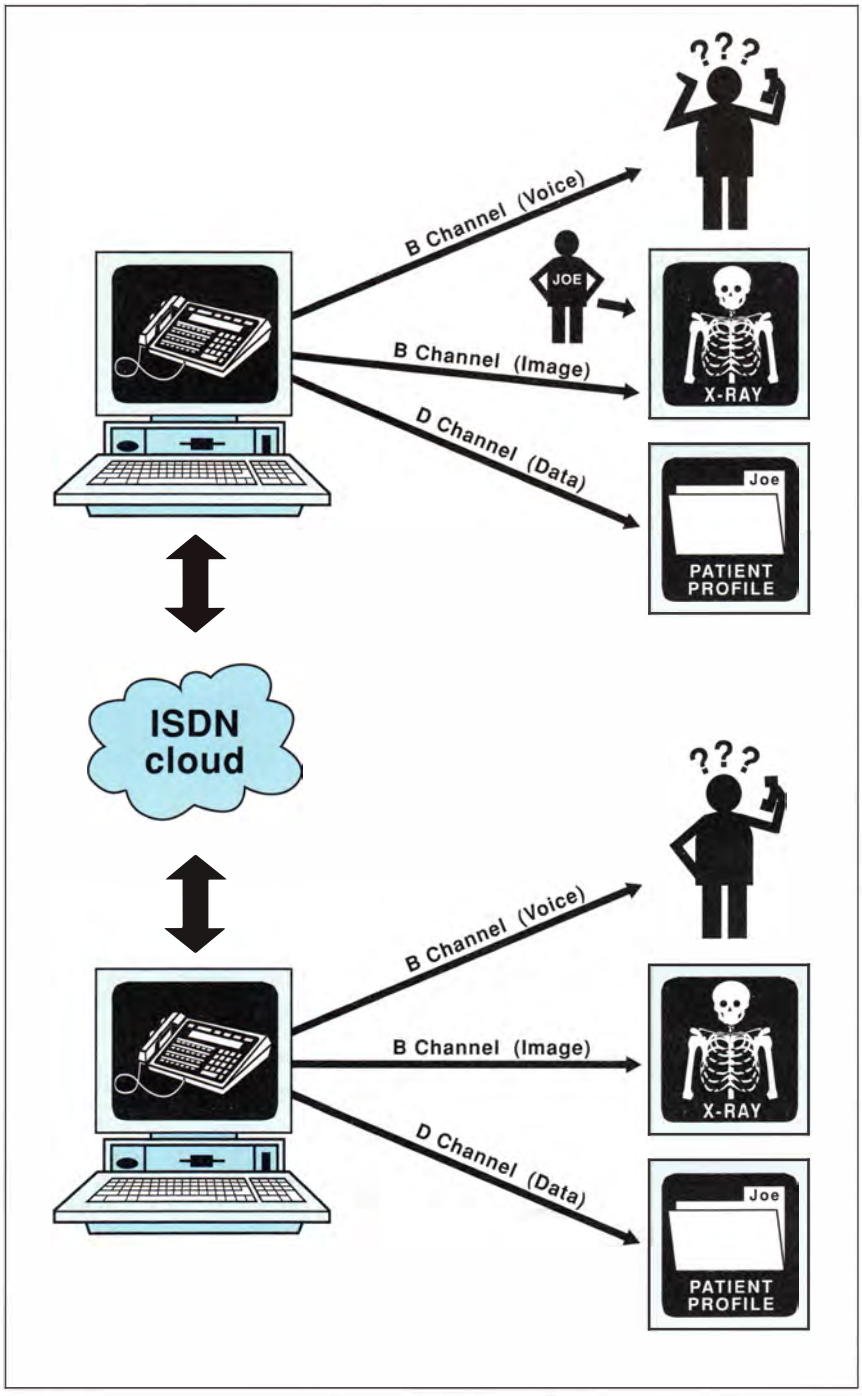


Figure 2. Medical Consulting Using ISDN

By using a toll-free telephone line and a computer as an answering machine, a user can receive customer orders 24 hours a day, worldwide. Add an expert system application and an unmanned computer may be able to answer a substantial percentage of customer inquiries, place customer orders, and perform other services that may have previously required the assistance of an operator or salesperson.

Team up ISDN with a network of computers at an elementary school, a few students in the library, and an on-line encyclopedia and the students' access to information is substantially increased. The encyclopedia could even be updated online, so that it would never go out of date. If Johnny is reading about caterpillars, Janie can still read in the same *volume* about cats. Maybe animation could be added to show the caterpillar turning into a butterfly.

The same ISDN network could be applied to computer-aided education. Multiple schools and school districts around the world could share educational resources such as "Writing to Read"®. School districts that never had access to these facilities could access them via the telephone. Self-study programs could be available to a wider population of students such as those with conflicting work schedules, those living far from a college, those embarrassed or uncomfortable in a classroom environment, those requiring language translation, or those who are bedridden or homebound. (See Figure 2.)

Easy access to computer databases may expand the services that chain stores can provide. The grocery chain may allow one to shop from home, do price comparisons, view the products offered within the store, place the order, and either pick up the supplies at a specified time or have them delivered to the front door. The grocery may consolidate these services into a central warehouse that stocks delicacies that are now financially feasible to provide. The drugstore chain may not only let a customer put in an order by phone, but also let them view their account history from all branches for the year. Over-the-counter medication might be included in this history to give the customer a more accurate record of what medicines he or she might have used.

Personal account data and/or passwords could be used to verify accounts and carry out financial transactions at the time of placing the order rather than sending checks and invoices.

These are just a sample of the applications possible as a result of ISDN's functionality, accessibility, and performance. ISDN has the potential of vastly increasing our access to information and services, and touching our day-to-day lives.

WHO WILL BENEFIT THE MOST FROM ISDN?

ISDN brings us two giant steps closer to a true information society: an electronically linked society depending less on printed information and the energy required to print and distribute that information; a society with integrated textual, pictorial, and audio information at its fingertips. Consider how the following ISDN applications might benefit both people and the Earth:

What if instead of sending paper greeting cards people sent animated electronic greetings?

How about an electronic city map? Always up to date, electronically finding a destination and highlighting the best route. It might even display pictures of landmarks along the route or point out major road construction or traffic tie-ups.

What if we looked up phone numbers in an electronic phonebook and we never put another phone book in a landfill again?

What if instead of recycling newspapers, newspapers were never manufactured in the first place?

What if you could browse the interiors of hotels before making a reservation?

What if you could work at home when your child was ill?

What if your child could still go to the library without leaving the house?

Businesses may also benefit financially with primary rate ISDN. Primary rate ISDN provides a switched high bandwidth medium for connecting different computers and networks. It can save businesses money by providing an economical alternative to leased T1 lines. The standardization that ISDN provides means that businesses have a greater base of suppliers from which to choose and lower costs handed down by switch and terminal equipment manufacturers.

WILL THE USER BE AWARE THAT HE/SHE IS USING ISDN?

When installing or configuring ISDN, there will be a number of new options and features available to customize your ISDN environment. However, if your system is already installed and configured, probably the only way you could tell that you were using ISDN would be if you noticed how fast data was being transmitted, or that there were several new features available.

WHAT IS IMPEDING THE USE OF ISDN?

There was, and still remains, skepticism about ISDN. The doubts are diminishing as digital networks are installed and supported around the world. Japan is now leading the world in installed networks, followed by Europe.

Commentators have suggested that there are a number of reasons why ISDN is not already a worldwide service. Many believe the Regional Bell Operating Companies (RBOCs) in the USA are slowly converting their local switches to digital. There are financial management and depreciation related considerations in going digital overnight — technology changes faster than the depreciation laws allow the equipment (switches) to change. Hence the decision to convert to a digital telephone switch becomes a financial one instead of a technical one. Independent equipment manufacturers (computer manufacturers, for example) are slow to offer ISDN equipment because they do not see a substantial market for their equipment due to RBOCs not offering ISDN services country wide. (This is referred to as the *islands of ISDN* problem.)

The regional public services commissions that regulate the RBOCs continue to discuss a tariff and rate structure for ISDN service. Experts believe this lack of a firm tariffing structure is one more reason why ISDN is not taking off as expected. One of the reasons why Japan and some European countries have ISDN deployed at a much larger scale than the USA is that they have attractive tariffing schemes that encourage customers to use ISDN services.

Until 1988 there were no solid ISDN standards. The CCITT standard addressed this. Different country implementations offer different subsets of the standard. Over time these will converge to one common set of offerings.

Some computer and data communications experts were dubious about whether, technically at least, ISDN really provided any advantage. For example, you could buy or lease special equipment from the telephone company that let you transmit data at 56 Kbps. Practically speaking, however, ISDN is much more convenient: ISDN user equipment like your PC plugs right into your regular telephone line. PCs with modems can plug into a phone line too, but even the more expensive modems only run at 19.2 Kbps, whereas with ISDN you get two 64 Kbps channels. Depending upon the tariffs, the difference in the data transmission speeds could reduce the phone charges. More and more Japanese and European customers are realizing these savings since they can transmit and receive data faster on ISDN, and hence save on connection charges.

Some people believe that what is needed to make ISDN really take off is a *killer application,* analagous to what the *spreadsheet application* did for the PC industry. Optimists believe these applications are coming — with particular potential in the multimedia, imaging, and database areas. All these applications transmit and receive a lot of data. With ISDN's higher speeds communication between computers running these application becomes feasible.

WHAT IS THE COST JUSTIFICATION FOR ISDN?

One of ISDN's goals is to provide relatively low cost transmission of data/voice at high speed rates. Cost is a key factor. Who will benefit from this cost reduction — just the person sending the data? No!

Public carriers will undoubtedly profit from ISDN. The maintenance cost of a digitized network will drop dramatically due to the shorter distance between the subscriber lines and the telephone exchanges.

Home or business users that use the telephone for both data and voice also benefit. ISDN's higher speed of transmission reduces the cost of sending data over a long distance. Data can be transferred over a single channel at 64 Kbps. This is higher than most existing transfer rates for WANs.

Tariffs need to be examined when estimating the cost of ISDN. A tariff is an additional cost to use the ISDN lines. If the tariff is too high, the cost advantage gained by sending data at a high speed is reduced. The exact cost structure to be used for ISDN services in U.S.A. has yet to be determined.

One area where ISDN is cost justified is *growth,* i.e., an increase in the number of people using a network or an increase in the volume of data on a network. For example, a 3270 LAN work group has 8 to 10 workstations doing database transactions on a 9.6 Kbps modem link to a host system. To add 10 more workstations, while maintaining the the same response time, they must add a second 9.6 Kbps modem, a new telephone and an extra port at the host system. ISDN adapter cards today cost about the same as a SDLC adapter card plus a 9.6 Kbps modem, but ISDN basic rate supports two 64 Kbps lines. If only one 64 Kbps line is needed to maintain workstation response time, the other is available for future growth.

The same idea holds true for an increase in data volume. If the same 8 to 10 workstations needed to transfer files to and from the host, the additional data volume would have a similar effect on workstation response time as adding additional workstations due to the

bottleneck associated with the slower line speed of modems. ISDN's superior throughput provides the capacity necessary to handle this increased load.

Other simple examples of ISDN's cost effectiveness are where long distance charges are based on the connect time. A GA facsimile using ISDN can send an A4 document containing 35,000 bytes of data, in four seconds or send a floppy disk file of 1.3 Mb in three minutes. The same disk file would take 40 minutes to send using a 2.4 Kbps modem.

WHY USE ISDN INSTEAD OF A MODEM?

The obvious reason is speed. A typical modem for a home computer has a throughput range of one to ten Kbps. With ISDN's Basic Rate Interface of two B-channels and a D-channel, the user's data rate is 144 Kbps. With the Primary Rate Interface, this soars to as high as 1544 Kbps.

The biggest advantage an ISDN connection has over a modem is that it supports multiple lines with multiple services. Many software packages for modems out in the market today provide for multiple sessions to the same host, but ISDN supports multiple hosts. Each B-channel operates as a completely separate phone line over which the user can dial a different phone number.

Finally, eliminating the modem means one less piece of hardware is necessary to plug into your WAN. This translates into better portability.

SUMMARY

Integrated Services Digital Networks are now provided by telephone companies worldwide. ISDN standards for connectivity to these telephone networks are defined by CCITT to allow for maximum flexibility in its usage while providing the structure to guarantee compatibility necessary to allow for global interconnection of diverse hardware and software applications. With ISDN comes the capability to simultaneously transmit voice and data over a single switched digital telephone line, and to transmit that data anywhere, cheaper and faster than could ever be done before.

Authors L-R: Katie Barrett, Bill Bulko, Denise Loose, Pat Scherer, Steve Kenney, and Pratik Nanavati. Missing from picture is Frank Grubbs.

Katie Barrett, *IBM Personal Systems LOB, 11400 Burnet Road, Austin, Texas 78758. She joined IBM in 1989. Previous to that she worked in a variety of fields, including symbolic processing, robotics, factory automation, communications, graphics and civil engineering applications. She was also on the faculty of the University of Kentucky. She received a BS degree in Computer Science from Pennsylvania State University and a MS degree in Computer Science from University of Kentucky.*

Frank Grubbs, *IBM Personal Systems LOB, 11400 Burnet Road, Austin, Texas 78758. He joined IBM in 1963 in the Atlanta, Georgia, branch office. In 1974 he transferred to Rochester, Minnesota, as the Communications Software Planner for the System/38. In 1980 he transferred to Austin Texas as the architecture manager for the 5280 system. Since being in Austin, he has held positions in a variety of areas: RTPC, related products, office system architecture and OS/2 EE development.*

Stephen Kenney, *IBM Personal System LOB, 11400 Burnet Road, Austin, Texas 78758. Mr. Stephen Kenney joined IBM in 1989. He received a BS in Computer Science and Engineering from Ohio State University and is currently working on an MBA at St. Edward's University.*

Pat Scherer, *IBM Personal Systems LOB, 11400 Burnet Road, Austin, Texas 78758. Ms. Scherer joined IBM in 1980 as an Industrial Engineer. She holds an MBA in Industrial Management and a BA in Microbiology from the University of North Texas and recently completed the Graduate Certificate Program in Computer Science from the University of Texas.*

Denise Loose, *IBM Personal Systems LOB, 11400 Burnet Road, Austin, Texas 78758. Ms. Loose joined IBM in 1985 and has worked on the IBM Voice Communications product and OS/2 Communications Manager. She received a BS in Applied Mathematics from Texas A&M University.*

William C. Bulko, *IBM Personal Systems LOB, 11400 Burnet Road, Austin, Texas 78758. Dr. Bulko joined IBM in 1989. He received a BS in Mathematics from the University of Texas at El Paso, an MS in the Computer Sciences from the University of Texas, and a PhD in the Computer Sciences from the University of Texas, specializing in artificial intelligence. His areas of expertise include natural language understanding, software architectures for AI, graphics, and user interfaces.*

Pratik Nanavati, *IBM Personal Systems LOB, 11400 Burnet Road, Austin Texas 78758. Mr. Nanavati joined IBM in October of 1987. He received his BS degree in Electrical Engineering from MS University, India, and is currently working on his MS degree in Computer Engineering at University of Texas at Austin.*

LAN Requester and Server

Disk Array Management Software From Integra Technologies, Inc.

by Wayne W. Wang and Wendi Nusbickel

This article discusses an OS/2® product that was first demonstrated at Fall COMDEX™, 1990 on the IBM® Future Vision stage.

BACKGROUND

High-end microcomputers have entered the workplace in large numbers worldwide. This huge installed base and the availability of powerful system software, such as OS/2 and UNIX®, have given rise to the use of high-end microcomputers in a wide range of applications previously handled by larger computers. These microcomputers need storage systems which have large capacity, faster data response time, and unfailing reliability.

The Disk Array Management Software is designed to fulfill those requirements with a minimum of hardware and software overhead. It is a product which manages an array of multiple Small Computer System Interface (SCSI) fixed disk drives, and provides data *striping* and fault tolerant protection as well as enhanced throughput.

Striping is a scheme that spreads data across all drives in the array. If any drive in the array fails, data still can be accessed through an error recovery procedure transparent to the OS/2 kernel and applications. However, the user is notified by a pop-up message on the screen stating that there is a specific drive down. The user can replace the DOWN drive and recreate the data at his or her convenience.

This software also enables the drives in the disk array to appear as a single large volume to the operating system. Then in all respects (even with a failed disk drive), the disk array is recognized by OS/2 as a virtual SCSI disk drive.

RAID LEVEL 5 APPROACH

With recent improvements in processor and memory speed, the I/O system has become the bottleneck in high performance personal computers. According to David A. Patterson, Garth Gibson, and Randy H. Katz, researchers at the University of California at Berkeley, "Increasing performance of CPUs and memories will be squandered if not matched by a similar performance increase in I/O." (See Patterson, et al., 1988). They phrase it as an "I/O Crisis". Those researchers provide solutions by introducing the concept of Redundant Arrays of Inexpensive Disks (RAID), which now is used commonly by the computer industry. They define five RAID schemes which provide data redundancy to increase reliability and provide parallel access to increase performance. Each scheme has its own characteristics. Further details on those schemes can be found in *Patterson, 1988.*

The Disk Array Management Software uses a RAID Level 5 approach, which is particularly suitable for online transaction processing applications. An example of this environment would be an OS/2 workstation on a Local Area Network (LAN) which accesses an OS/2 database server. These kinds of applications perform I/O as small read/modify/write operations.

Wayne W. Wang

Wendi Nusbickel

If a drive fails, data can still be accessed

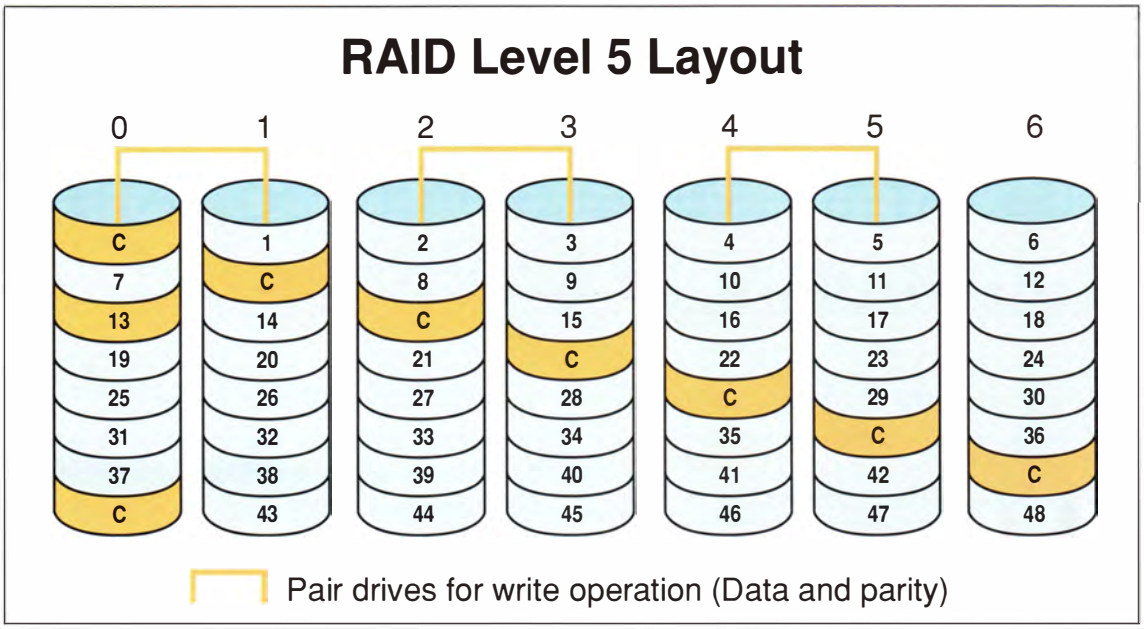


Figure 1. RAID Level 5 Layout

RAID Level 5 stripes data and error correction code across all drives. It also sustains both parallel read and write operations among disk drives in the array. Figure 1 illustrates the layout of disk drives for RAID level 5.

1. C indicates ECC (Error Correction Code), parity, or check block. Logical data blocks are striped across all drives, as indicated by the numbers.

2. This layout, with seven hard disk drives, can support a maximum of seven simultaneous READ operations (when the virtual I/O blocks are located on different physical drives). However, real throughput of these READ operations may be limited by the SCSI bus bandwidth since all fixed disks share the same SCSI bus.

3. This layout can support a maximum of three simultaneous WRITE operations (when the pair of virtual data and parity blocks are on different physical drives.) Each WRITE operation needs to update the data and parity drives (refer to Patterson 88). Real throughput of these WRITE operations also may be limited by the the SCSI bus bandwidth since all fixed disks share the same SCSI bus.

Software/Hardware Platform

To run the Disk Array Management Software for OS/2, OS/2 version 1.30.1 is required. This is because a disk identifier for disk arrays was added to this release of OS/2. It indicates that a drive is part of a disk array and is not managed by the OS/2 base disk device driver.

The hardware required (beyond what is required for OS/2) is a 386™/486™-based PS/2® and an IBM SCSI adapter (cached or non-cached). Connected to this SCSI adapter is the "disk array", which consists of 3 to 7 IBM SCSI fixed disks (the IBM SCSI 160Mb, 320Mb or 400Mb drives). These drives could be contained in a PS/2 external enclosure for SCSI devices, multiple PS/2 SCSI Storage Enclosures, or (depending on the number of drives in the array), a PS/2 Model 95.

Software Disk Array Solution

Unlike many other disk array products currently available in the marketplace, this is a software solution. It works on existing IBM hardware, so the customer that wants a fault tolerant disk array need not make a big investment in a costly hardware disk array product.

Functions

The Disk Array Management Software for OS/2 contains an installable OS/2 fault tolerant SCSI disk device driver, and utility programs which help the user to manage the disk array.

Fault-tolerant SCSI Disk Device Driver

This OS/2 installable device driver supports the standard OS/2 device driver interface. The strategy entry point is called by an I/O request thread with an I/O request packet.

Path For Standard I/O Requests

The standard I/O request packet sent through the strategy entry point will be processed and blocked at task time. The I/O thread running at task time will not return to the OS/2 kernel until the I/O operation has completed. Depending on the drive status in the disk array, the I/O thread performs the following steps at task time.

1. Converts the OS/2 standard I/O request format into an I/O format that is independent of the operating system.

2. Transforms the logical request sector number into a physical drive and sector number.

3. Starts the physical I/O requests to the hardware.

4. Blocks the I/O thread until the physical I/O requests are completed.

5. If the I/O thread wakes up with no errors, it then steps through the state machine. If more I/O requests are ready, then it returns to step 3. Otherwise, it returns to the OS/2 kernel with a "successful" status.

6. If the I/O thread wakes up with an error, it then steps through the error recovery procedure. Then, if more physical I/O requests are ready, it returns to step 3.

With the multitasking support of OS/2, the disk array device driver can access multiple drives and operate multiple outstanding I/O requests, thereby taking maximum advantage of the RAID Level 5 architecture.

All utility programs communicate with the disk array driver through vendor unique IOCTLs. Most of the Disk Array Management Software utility programs are designed to run in background with lowest priority, so normal I/O operations will not be disturbed.

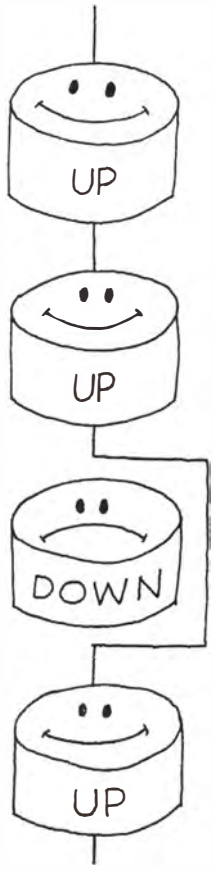
The Daemon Program

MONITOR.EXE, the daemon program running in the background, is invoked at system start-up time by a "RUN=" statement in the OS/2 C:\CONFIG.SYS file. This program is very critical to the entire disk array. It monitors disk array status periodically, reports any error messages to the user, and records disk array information to the disk array log file.

An important feature of the Disk Array Management Software is error transparency to the operating system. All disk array error recovery procedures are handled by the disk array software. If any drive in the disk array fails, neither the OS/2 kernel nor any user application knows that there is anything wrong with the disk array. Obviously, however, the system administrator should be notified of any problems with the disk array, and the Monitor program is equipped to fulfill this requirement. Since MONITOR is a background task, all error messages reported to the user are invoked by pop-up messages on the screen. These messages also are logged into the disk array log file.

The Format Program

The PACFMT program is the format utility for the disk array. It formats the disk array with options such as whether the array is to appear as a single or as multiple volumes, what file system the array will contain (FAT or HPFS), the array volume label(s), and the ECC striping units. The ECC striping unit or blocking size defaults to one physical cylinder spread across each disk in the data-stripe (refer to Figure 1). This program also initializes all the drives in the array with data striping information.



ECC Verification and Correction Program

Redundant data, also known as check data, error correction code (ECC), or parity, are striped across all drives. These redundant data occupy the space of one drive in the array and give the Disk Array Management Software the ability to tolerate any failure of a single drive in the disk array. The PACSCAN program verifies that the check data are correct. It scans the check data cylinder by cylinder (if there is no outstanding I/O). If the check data have been corrupted (for example due to a system power failure during a write operation to the array), the program will update the check data to match the data on the array. It is recommended that the system administrator run this program periodically to ensure the integrity of the check data. This program runs as a background task with lowest priority.

```
char    paclog[] = "C:\\INTEGRAI\\PACLOG.EXE" ; /* Log file name */
char    envlist[1024];          /* environment ptr                */
char  * myenvp [60]  ;          /* array of pointers to environ.  */
/*****************************************************************/
/*                                                               */
/* invoke_logging()                                              */
/*                                                               */
/* This function invokes another program 'PACLOG.EXE' to write   */
/* event information to the INTEGRA.LOG file.                    */
/*****************************************************************/
void    invoke_logging( array_num, drive_num, sector_num, command)
USHORT  array_num ;             /* disk array number             */
USHORT  drive_num ;             /* DOWN drive number             */
ULONG   sector_num;             /* sector number in drive        */
USHORT  command   ;             /* command to invoke logging     */
{
  char    arglist[400];         /* argument list                 */
  char    ONBuf[400]  ;         /* fail object name buffer       */

  /* Set up the argument list from the input parameters.         */

  /* Also get the current environment and convert it to DosExecPgm*/
  /* format. This is so the called program has access to any     */
  /* environment variables it may need.                          */
      .
      .
      .
result = DosExecPgm( ONBuf,          /* fail object name buffer  */
                     sizeof(ONBuf), /* length of object name buf*/
                     EXEC_ASYNC,    /* execution flag - async   */
                     arglist,       /* argument list            */
                     envlist,       /* environment list         */
                     &pid,          /* child process ID         */
                     paclog);       /* program name             */

/* if PACLOG.EXE not found                                       */
if ( result == 2 )
    printf( rcrt_msg[ FILE_NOT_FOUND ], paclog );
}
```

Figure 2. Listing of Invoke Logging

Data Regeneration Program

Although the system still runs with a failed disk drive, it runs in degraded mode. Degraded mode means that the disk array device driver must "reconstruct" the data that was contained on the DOWN drive by obtaining data and fault tolerant parity from the remaining drives in the array. This drive "reconstruction" requires additional overhead in the system, so response to the disk array will be slower than when all drives are UP.

Having a drive DOWN in the disk array also means that it is operating without fault tolerance. No fault tolerance means that if another drive in the array should fail, data will be lost. To prevent loss of data, failed drives should be replaced, and recreated with the RECREATE program, as soon as possible. This program will recreate data back onto the replaced drive cylinder-by-cylinder, and runs as a background task with lowest priority.

It is important to note that users can access the disk array while a drive is being recreated. This means that the system is DOWN only as long as it takes to install a new drive. This results in high data availability, even in the middle of the data regeneration process.

Logging Program with the Option of Customization

A logging program is supplied to record all critical events. Each record has the following fields: array number, drive number, sector number, event description and time stamp. The logging program is started by other utility programs which use the OS/2 DosExecPgm() call to pass event arguments.

In some working environments a network server running the Disk Array Management Software may be locked up or accessible only to the network system administrator. The result is that pop-up messages on the system console may not be seen for some time. It is best to be aware of any potential problems as soon as possible so that corrective action can be taken. Therefore, the source code of this logging program is supplied with the intent that users may customize it to their environments.

For example, in an OS/2 LAN client-server environment, it was desirable to send a network message to a specific user on the LAN whenever a disk array event was logged into the INTEGRA.LOG file. In this case, the PACLOG.C program was modified to utilize the network API NetMessageBufferSend().

```
/*****************************************************************/
/*                                                               */
/* function: main                                                */
/*                                                               */
/* This program is invoked by INTEGRA utility programs to log the */
/* events to INTEGRA.LOG file.  The source is supplied to the user*/
/* for customization.  The PACLOG.EXE program has to be resident  */
/* in C:\INTEGRAI directory. There are five arguments passed to   */
/* this program:                                                 */
/*          Array Number  -  Integer                             */
/*          Drive Number  -  Integer                             */
/*          Sector Number -  long                                */
/*          Event Number  -  Integer                             */
/*          Event description text string                        */
/*                                                               */
/*****************************************************************/
main(argc, argv)
int      argc;
char    *argv[];
```

Figure 3. Listing of PACLOG.C (Continued)

```
    {
    USHORT      rc;
    CHAR        temp_buf[100];
    CHAR        *s_ptr, *d_ptr;

    /* Program variables are defined and initialized...          */
    . . .

    array_number  = atoi( argv[ 1 ] );/* 1st arg is array number  */
    drive_number  = atoi( argv[ 2 ] );/* 2nd arg is drive number  */
    sector_number = atol( argv[ 3 ] );/* 3rd arg is sector number */
    command       = atoi( argv[ 4 ] );/* 4th arg is command number*/
    strcpy( event_strptr, argv[ 5 ] );/* 5th arg is event text    */

    /* process log message and output to log file                */
       . . .

    /* Begin customization sample code*****************************/

    /* The following sample code is compiled into PACLOG.EXE.    */

    /* Set up the event message string.                          */
    strcpy( temp_buf, array_str);

    /*Add the drive number if appropriate.                       */
    if(drive_number != NO_DRIVE)
       strcat( temp_buf, drive_str);

    /* Add the sector number if appropriate.                     */
    if(sector_number != NO_SECTOR)
      strcat( temp_buf, sector_str);

    /* Add this information to the end of the event text string. */
    strcat(event_strptr,temp_buf);

    /* Get the name of the administrator workstation the user has */
    /* set in the environment (if it exists), and send a network  */
    /* message to this workstation on the LAN.  To receive this   */
    /* message as a pop-up on the console, the administrator      */
    /* workstation starts the network services of MESSENGER and   */
    /* NETPOPUP.                                                  */
    if( (s_ptr=getenv("INTEGRA_MSG_NAME") ) )
       NetMessageBufferSend ( (const char far *)0L,
                              (char far *)     s_ptr,
                              (char far *)     event_strptr,
                              (unsigned short)
strlen(event_strptr));
    /* End customization sample code*******************************/
    exit(0);
    }
```

Figure 3. Listing of PACLOG.C

Now, whenever an event is logged, the user specified in the call to this API will receive a notification of the logged condition. The only additional requirement on the user system was that it start the MESSENGER service. This is particularly useful for notifying a network administrator of any disk array error conditions.

Performance

According to a paper titled "*Introduction to Redundant Arrays of Inexpensive Disks (RAID)*" Patterson, et al., 1989), the expected performance overhead for RAID 5 write operations is as follows:

"For large writes (writing at least a sector to every disk in the parity group) the only performance hit is 1/N more writes to write parity information. Writes to data on a single disk, on the other hand, require four disk accesses:

1) Read the old data

2) Read the old parity

3) Write the new data

4) Write the new parity using this formula:

new parity = (old data XOR new data) XOR old parity."

In the personal computer environment of the Disk Array Management Software, almost all writes are considered small, or writes to data on a single disk as just described. This is because for personal computers based on the Intel 80x86 architecture, I/O data is limited by a segment size of 64Kb. Additionally, the data packet sizes may be even smaller when network communications and cache management are considered. Therefore, a RAID-5 write operation in this environment always takes the penalty of four disk accesses, which can only be compensated by parallel I/O to the different drives in the disk array.

Another significant difference between the operating environments of the Disk Array Management Software and papers by Patterson, et al., is the number of drives in the disk array. Since those papers are based on a mainframe environment, the number of drives in their disk array environment is in the hundreds. Therefore, the overall parallelism should compensate well for the overhead of a write operation.

Benchmark Results

Tests were run comparing the data transfer rate (or throughput) of the disk array managed by the Disk Array Management Software with a single drive being managed by the standard OS/2 disk device driver. Both the disk array and the single drive were attached to the same server system.

The results in Figure 4 show benchmarks that were run from five workstations accessing an OS/2 LAN Server running OS/2 Extended Edition 1.30.1 with OS/2 LAN Server 1.30.1. The server hardware was a PS/2 Model 95 system with a 33 MHz processor and 16Mb of memory. The PS/2 workstations were connected to the server on a 16mb/s Token Ring Local Area Network (LAN) and were running OS/2 Extended Edition 1.3. The disk array consisted of six 400Mb IBM SCSI fixed disk drives. The single drive tests also were performed on a 400Mb IBM SCSI drive.

Tests were performed in three categories: random reads, random writes, and random I/O consisting of 3 reads followed by a random write (simulating an application environment). The block size used for transfers was 4Kb; the size of the file being accessed was 5Mb.

Performance Summary

As mentioned, small WRITE operations of RAID 5 have significant overhead compared to normal operations, and those overheads shown on the benchmark write tests are exactly as expected. However, in the real world application environment, the number of READ requests usually exceeds the number of WRITE requests. In this case the disk array has much better performance than a single disk, as shown on the random I/O tests of 3 reads and 1 write.

Note that the estimated overhead within the driver to generate new parity on these benchmarks ranged from approximately 1.5 to 2.8 times.

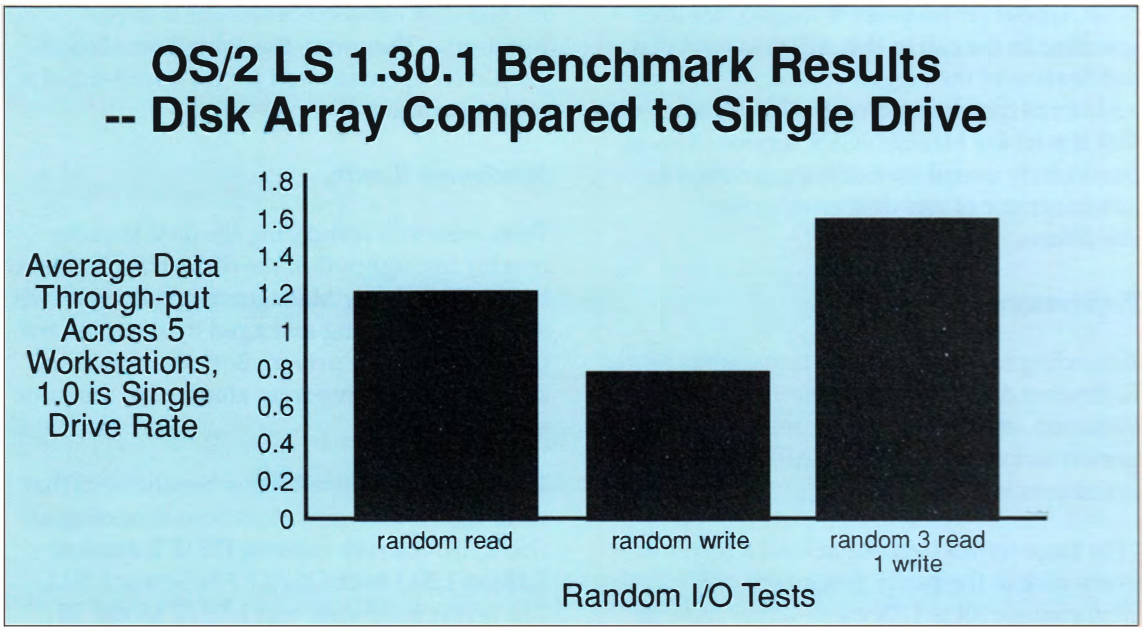


Figure 4. Benchmark Results

FUTURE

Areas of enhancement being considered for the Disk Array Management Software include: supporting a hybrid of different RAID levels, an option to disable parity for a performance gain, implementing multiple I/O channels in a single disk array, supporting multiple disk arrays in a single system and allowing more than seven drives in a disk array.

CONCLUSION

The Disk Array Management Software for OS/2 from Integra Technologies, Inc., is a cost effective solution that provides fault tolerance with OS/2 transparency and increased logical volume capacity, and with minimal performance overhead.

REFERENCES

Patterson, D. A., G. Gibson, and R.H. Katz, *A Case for Redundant Arrays of Inexpensive Disks (RAID)*, ACM SIGMOD Conference, Chicago, IL, (June 1988).

Patterson, D. A., P. Chen, G. Gibson, and R.H. Katz, *Introduction to Redundant Arrays of Inexpensive Disks (RAID)*, Spring COMPCON 89, March 1, 1989

Schulze, M., G. Gibson, R.H. Katz, and D.A. Patterson, *How Reliable is a RAID?*, Spring COMPCON 89, March 1, 1989

Wayne W. Wang, *Integra Technologies, Inc., 3130 De La Cruz Blvd. Suite 200, Santa Clara, CA 95054 (408) 980-1371. Mr. Wang is a Software Project Manager and has been responsible for software design and implementation for variety of disk array subsystems since 1988. Mr. Wang holds a BS from National Cheng Kung University, Taiwan and an MS in Computer Science from Central Michigan University, Michigan.*

Wendi Nusbickel, *IBM Entry Systems Division, 1000 NW 51st Street, Boca Raton, FL 33431. Ms. Nusbickel joined IBM Boca Raton in 1983, and is a programmer on the OS/2 Servers and Special Projects team. She has a BS in Computer Science from North Dakota State University in Fargo, North Dakota.*

International Business Machines Corporation
P.O. Box 1328
Boca Raton, FL 33429-1328

G362-0001-11